ENABLING TECHNOLOGIES FOR REAL-TIME SIMULATION

University of Massachusetts
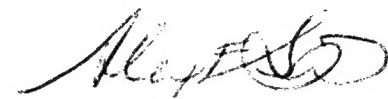
Christos G. Cassandras and Wei-Bo Gong

19971126 138

DTIC QUALITY INSPECTED 3

**Rome Laboratory
Air Force Materiel Command
Rome, New York**

This report has been reviewed by the Rome Laboratory Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.
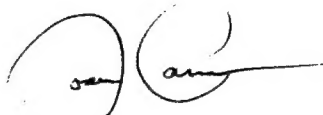
RL-TR-97-77 has been reviewed and is approved for publication.

APPROVED:

ALEX F. SISTI
Project Engineer

FOR THE DIRECTOR:

JOSEPH CAMERA, Technical Director
Intelligence & Reconnaissance Directorate

REPORT DOCUMENTATION PAGE

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	Aug 97	Final Sep 95 - Dec 96

4. TITLE AND SUBTITLE

ENABLING TECHNOLOGIES FOR REAL-TIME SIMULATION

6. AUTHOR(S)

Christos G. Cassandras and Wei-Bo Gong

5. FUNDING NUMBERS

C - F30602-95-C-0242
PE- 62702F
PR- 4594
TA- 15
WU-N3

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

University of Massachusetts
Dept of Electrical and Computer Engineering
Amherst MA 01003

8. PERFORMING ORGANIZATION REPORT NUMBER

N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Rome Laboratory/IRAE
32 Hangar Rd
Rome NY 13441-4114

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

RL-TR-97-77

11. SUPPLEMENTARY NOTES

Rome Laboratory Project Engineer: Alex F. Sisti/IRAE/(315) 330-3983

12a. DISTRIBUTION AVAILABILITY STATEMENT

Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This report summarizes the work performed for "Enabling Technologies for Real-Time Simulation". The objective was to develop and study a three-pronged approach for turning simulation into an effective setting for C3I application areas. In particular, the report describes research into 1) concurrent simulation, for speeding up simulation experiments; 2) hierarchical simulation; and specifically, preserving statistical fidelity in hierarchical decomposition activities, and 3) using Neural Networks as a method to extract a metamodel from the simulation.

14. SUBJECT TERMS

Concurrent Simulation, Hierarchical Simulation, Statistical Fidelity, Metamodel, Model Abstraction

15. NUMBER OF PAGES

108

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

TABLE OF CONTENTS

1. INTRODUCTION.

This report summarizes the work we have performed for the project entitled "Enabling Technologies for Real-Time Simulation". The objective of this effort has been to develop and study a three-pronged approach for turning simulation into an effective setting for C^3I application areas. In particular,

(a) Using *Concurrent Simulation* techniques, the goal is to extract as much information as possible from one simulation; this is by itself an obvious desired goal, but it will also provide this key capability as a supporting component for the other two directions.

(b) Hierarchical decomposition is one way to reduce complexity encountered in simulation models of interest; the challenge is to do it while preserving statistical fidelity, which we attempt to accomplish by means of a *Path Bundle Grouping* (PBG) approach, related to the theory of cluster analysis.

(c) Extracting a *metamodel* from the simulation process is a second way to reduce complexity; again, the challenge is accuracy and our approach is aimed at exploiting the use of *Neural Networks* as general-purpose modeling devices towards this goal.

The scope of the project has been to develop specific algorithms and prototype software demonstrations of our approach in specific C^3I application areas. Thus, appropriate simulation models were built, and algorithms based on the proposed new techniques were developed and tested.

In what follows, so as to place our work in the proper context, we first identify some of the major challenges faced by simulation technology today and the approaches we are following to address these challenges (Section 1.1). We then describe the organization of this report (Section 1.2).

1.1. Issues in Discrete Event Simulation.

Simulation is widely recognized as one of the most versatile and general-purpose tools available today for modeling complex processes and solving problems in design, performance evaluation, decision making, and planning. This includes C^3I environments, where most problems confronted by designers and decision makers are of such complexity that their analysis and solution far surpass the scope of available analytical and numerical methods; this leaves simulation as the only alternative of "universal" applicability.

The importance of discrete event simulation has given rise to a number of commercially available software packages (e.g., SIMAN, SLAM, SIMULA, SIMSCRIPT, MODSIM, EXTEND) whose applicability ranges from very generic to highly specialized. However, the use of typical simulation software is limited by factors such as the following: (a) One must have

thorough knowledge of the specific tool at a detailed technical level before attempting to use it in a modeling effort. (b) One must be an experienced programmer, in addition to a decision maker. (c) In order to make decisions based on simulation, one usually needs to run a large number of simulations and then carefully manage all output data collected on a case-by-case basis. (d) The field of simulation was developed primarily as a special branch of statistics involving dynamical phenomena. Manual handling and analysis of input/output data is still the norm, while design of interfaces, componentware interoperability, intelligent and automated analysis of output have been neglected. For example, the practice of Object Oriented Programming (OOP), with few exceptions, is still nascent in simulation languages despite the fact that the OOP idea actually originated in simulation. In addition, hardware advances, such as massively parallel computers and workstation networking, are only beginning to be noticed in simulation theory and practice. (e) The ultimate purpose of simulation is often system performance evaluation and optimization. However, simulation is notoriously computer time-consuming when it comes to parametric studies of system performance. Unless substantial speedup of the performance evaluation process can be achieved, systematic performance studies of most real-world problems are beyond reach, even with supercomputers.

With this brief discussion in mind, we identify below some of the issues that we believe constitute the major challenges faced by simulation technology today, and introduce some key ideas which have been the subject of further study in this project.

1. *"What if" capability and concurrent* simulation. A major goal of any simulator is to provide the capability to explore a multitude of "what if" scenaria. The obvious way to obtain answers to N "what if" questions is to perform $(N+1)$ separate simulations: one for some baseline scenario and N additional ones for each "what if" question. If a typical simulation run takes T time units, this procedure requires a total of $(N+1)T$ time units. In *concurrent* simulation, this objective is met by performing a single baseline simulation, but endowing it with the capability to generate all $(N+1)$ desired simulations concurrently. This is accomplished by exploiting an intelligent sharing of data which results in a total simulation time of $(T+c) << (N+1)T$, where c represents the overhead corresponding to this "intelligent data sharing". Concurrent simulation can be carried out on any sequential computer. While the feasibility of performing concurrent simulations has been established, the universal applicability of this approach remains an open issue and is one of the main components of this report.

2. *Hierarchical simulation and statistical fidelity*. Many simulation models, and combat simulation models in particular, have a distinct hierarchical structure. The lower, high resolution, level simulator generates reports which are then taken as inputs for the higher level simulator. Current practice is to use the mean values of variables from the lower level reports as the input to the higher level. This implies that significant statistical information (i.e., *statistical*

fidelity) is lost in this process, resulting in potentially completely inaccurate results. Especially when the ultimate output of the simulation process is of the form 0 or 1 (e.g., "lose" or "win" a combat), such errors can provide the exact opposite of the real output. Our effort has been directed at developing an interface between the two simulation levels to preserve the statistics to the maximum extent that the available computing power allows. Our work has shown that a "digital diffusion network" can serve such purpose well. The basic idea of the diffusion network is to take the lower level simulator reports as "training examples". These are data used to train the network so that it behaves as a *surrogate model* for the lower level simulator. The resulting network will provide outputs whose statistics should match those generated by the original lower level simulator. Precise systematic ways for accomplishing this goal remain a challenge, especially given the large dimensionality of the data involved. An approach further described in this report is based on the *Path Bundle Grouping* (PBG) approach, related to the theory of cluster analysis.

3. *Metamodeling through Neural Networks.* The main idea of metamodeling is to extract as much information from simulation as possible and process it so as to build a *surrogate model* of the system of interest which is much simpler (yet accurate) to work with. This is essentially analogous to constructing a function $F(x_1,...,x_N)$ from only selected values observed under selected combinations of values of $x_1,...,x_N$. The problem, of course, is that the actual function we are trying to approximate with $F(x_1,...,x_N)$ is unknown. The most common approach is to try and build a *polynomial* expression. This is often inadequate because if the shape of the actual curve corresponding to $F(x_1,...,x_N)$ includes sudden jumps and asymptotic behavior (which is very often the case from experience), then polynomial fits to such curves are known to be poor. Thus, obtaining a metamodeling device of "universal" applicability, i.e., one capable of generating functions of virtually arbitrary complexity, remains an open issue. This project explores *neural networks* as offering this capability and is intended to investigate the advantages and limitations of this approach.

1.2. Report Organization.

The contents of this report may be outlined as follows.

- **Section 2:** The basic theoretical framework for explaining the concurrent simulation algorithms we have developed is first presented. Based on this framework, a general concurrent simulation approach is introduced and a detailed *Time Warping Algorithm* (TWA) is described. The concept of "speedup" is also introduced in order to provide a clear quantitative measure of the improvement provided by concurrent simulation over conventional simulation techniques. Using a "speedup factor" metric, a number of detailed numerical examples are provided in this section.

- **Section 3:** Neural networks as universal function approximators are introduced. A particular type of neural network, the *Cascade Correlation Neural Network* (CCNN), has been identified for the purposes of this project for its ability to build itself to an appropriate size as part of its learning process during the training phase. The specific training algorithm we have used is described in detail in this section.

- **Section 4:** A testbed model using the *Tactical Electronic Reconnaissance Simulator* (TERSM) supplied by the Rome Laboratory is reviewed. This model has been used by previous researchers to develop and evaluate polynomial metamodels. As part of this project, we have reproduced polynomial metamodeling results reported in the literature as a baseline for evaluating the new methods we have developed using the CCNN presented in Section 3. Detailed comparative numerical results are provided for a problem previously reported in the literature demonstrating the advantages of the CCNN metamodeling approach. A new, more challenging, TERSM-related problem is also formulated and comparative results based on both approaches are provided.

- **Section 5:** A concrete hierarchical battle simulation model is introduced and the stochastic fidelity preservation issue is discussed in detail. A mathematical review of the model is presented and the *Path Bundle Grouping* (PBG) approach as a way to maintain stochastic fidelity in hierarchical simulations is subsequently described with several numerical results included. The PBG approach is related to the theory of cluster analysis and in this section we describe how the Adaptive Resonance Theory (ART) neural network can be employed to automate the task of path bundle grouping.

- **Section 6:** A new benchmark model is proposed in this section, motivated by the fact that the TERSM environment lacks certain challenging features that make metamodeling particularly difficult. This model is based on an *Aircraft Refueling and Maintenance System* (ARMS). We describe the ARMS model and present some results to demonstrate its features and complexity.

- **Section 7:** We present the main conclusions of our study, including lessons learned and recommendations. We also outline our ongoing work and some future research directions.

2. CONCURRENT SIMULATION.

As already mentioned in Section 1, a major objective of this project is motivated by the time-consuming nature of system performance exploration through simulation: to obtain answers to N "what if" questions, $(N+1)$ simulations are needed. Therefore, our goal is the following: *From a single simulation, obtain answers to all N "what if" questions simultaneously.* The main idea behind the approach we used to solve this problem is to observe the evolution of a single sample path of the Discrete Event System (DES) under study, called the *nominal* sample path, as it operates under some parameter. As the sample path evolves, observed data (e.g., event occurrences and their corresponding occurrence times) are processed to concurrently construct the set of sample paths that would have resulted if the system had operated under a set of different (hypothetical) parameters. Using these "concurrently constructed" hypothetical sample paths, it is possible to "concurrently estimate" the corresponding performance measures which can be used in the design of the actual system.

In this section we explain the principles of concurrent simulation. We begin in Section 2.1 describing what is known as the "sample path constructability problem". In Section 2.2 we review the concept of a stochastic timed state automaton as a modeling framework for general DES, and go on to describe the general procedure for constructing sample paths of DES. In Section 2.3 we present concurrent simulation as a solution to the sample path constructability problem. A method for quantitatively comparing the efficiency of concurrent simulation to "brute force" simulation is described in Section 2.4. The concept of "speedup factor" is presented as the basic measure of efficiency, and we identify some of the fundamental bounds this measure satisfies. Results are presented in Section 2.5 to demonstrate the benefits of "concurrent simulation," and we conclude with Section 2.6 where we briefly summarize the work done and address issues for future research. Finally, in an Appendix to this Final Report we have included a copy of the paper "Concurrent Sample Path Analysis of Discrete Event Systems" (recently published in the *Proceedings of the 35th IEEE Conference on Decision and Control*), which presents the "concurrent simulation" algorithm in a more rigorous way; in addition it includes some other applications of the algorithm (e.g. in *on-line* optimization).

2.1. Sample Path Constructability.

Consider a DES and a finite discrete parameter set $\Theta = \{\theta_1, \ldots, \theta_m\}$, where each parameter $\theta_j \in \Theta$, $j = 1, \ldots, m$ is in general vector-valued. Suppose the sample path generated by the DES is a function of the parameter θ_j (e.g., number of buffer slots in a queueing system), and designate the sample path generated under parameter θ_j by the sequence of pairs $\{e_k^j, t_k^j\}$, where $k = 1, 2, \ldots$ is an event-counting index, e_k is the k-th event (e.g., job arrival or departure), and t_k is the

occurrence time of the k-th event (equivalently, a sample path can be defined by $\{x_k^j, t_k^j\}$, $k = 1,2,\ldots$, where x_k is the state entered when the k-th event occurs at time t_k). Now, assume that the DES is operating under θ_1 and that all events and event times e_k^1, t_k^1 for $k = 1,2,\ldots$, are directly observable. The problem, then, is to use the observations of the sample path $\{e_k^1, t_k^1\}$ to construct the sample paths $\{e_k^j, t_k^j\}$, $k = 1,2,\ldots$, for any θ_j, $j = 2,\ldots,m$, as shown in **Fig. 2.1**. This problem is referred to as the *sample path constructability problem* [6]. Ideally, we would like this construction to be done concurrently so that at any point during a lengthy simulation process we can obtain estimates from all sample paths $j = 2,\ldots,m$, and, most importantly, we would like the time needed to obtain the concurrently generated estimates to be less than the time required for m individual simulations.
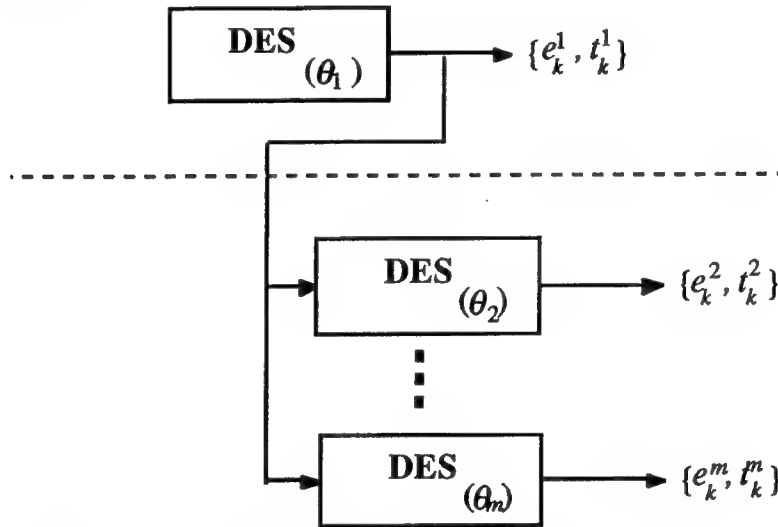
Figure 2.1. The sample path constructability problem.

Any sample performance metric $L(\theta_j)$ (e.g., the average processing time of a job for some buffer capacity) is obtained as a function of the corresponding sample path $\{e_k^j, t_k^j\}$, $k = 1,2,\ldots$ The importance of the sample path constructability problem, therefore, becomes clear when placed in the context of the following basic optimization problem:

(P) find $\theta \in \Theta$ to minimize $J(\theta) = E[L(\theta)]$

where we are careful to distinguish between $L(\theta)$, the performance obtained over a *specific sample path* of the system and $J(\theta)$, the *expectation over all possible sample paths*. The solution to the sample path constructability problem, if it exists, enables us to learn about the behavior of a DES under all possible parameter values in Θ from a single "trial", i.e., a single sample path obtained under one parameter value.

2.2. Stochastic Timed State Automata.

To explain the principles of concurrent sample path construction and estimation, it is useful to review how a single sample path is formally constructed for any DES. To do this we will make use of the *stochastic timed state automaton* (e.g., see [2]), which provides a general framework for modeling DES.

We begin by reviewing the concept of a *state automaton*. A *state automaton* is defined by (E, X, Γ, f, x_0), where E is a countable *event set*, X is a countable *state space*, $\Gamma(x)$ is a set of *feasible* or *enabled* events, f is a *state transition function*, and $x_0 \in X$ is an *initial state*. The feasible event set $\Gamma(x) \subseteq E$ and is defined for all states $x \in X$. The feasible event set reflects the fact that it is not always physically possible for some events to occur. The state transition function $f(x,i), f: X \times E \to X$, is defined only for the feasible events $i \in \Gamma(x)$; it is undefined for $i \notin \Gamma(x)$. It is also possible to replace the state transition function f by a state transition probability function $p(x'; x, i)$ representing the probability that the next state is x' given that the current state is x when event i occurs.

A *timed* state automaton $(E, X, \Gamma, f, x_0, V)$ is obtained when the model above is endowed with a *clock structure*, $V = \{v_i, i \in E\}$. This clock structure associates with every event $i \in E$ a real-valued clock sequence $v_i = \{v_i(1), v_i(2), \ldots\}$, where, $v_i(j)$ is the j-th *lifetime* of event i. The j-th lifetime is the amount of time between the instant when this event is enabled for the j-th time and its next occurrence.

Finally, in a *stochastic* timed state automaton $(E, X, \Gamma, f, x_0, G)$, the clock structure V is replaced by a set of probability distribution functions $G = \{G_i, i \in E\}$. In this case, the clock sequences $v_i = \{v_i(1), v_i(2), \ldots\}$ are random processes. For simplicity, we usually assume that the lifetimes $v_i(1), v_i(2), \ldots$ are i.i.d. random variables with distribution G_i. Thus, to generate a sample path of the system, whenever a lifetime for event i is needed we obtain a sample from G_i. The state sequence generated through this mechanism is a stochastic process known as a *Generalized Semi-Markov Process* (GSMP) (see also [12, 14]).

It will be helpful later on if we summarize the exact steps involved in generating a sample path of a stochastic timed state automaton. In addition to the state x of the underlying automaton, let us define two more state variables as follows. First, let us associate with each feasible event a state variable τ_i to denote the next occurrence time of event i. The usefulness of this variable, is that, given the occurrence times for each feasible event, the next event to occur, called the *triggering event*, is given by

$$e' = \arg\min_{i \in \Gamma(x)} \{\tau_i\}$$

and the time at which the event occurs is immediately given by $t' = \tau_{e'}$. The other state variable which we will find useful is the *score* of event i which we will denote by $s_{i,n}$: After n total events have been observed in a sample path, the score $s_{i,n}$ is the number of events of type $i \in E$ that have occurred. In the following subsection we describe a scheme that can be used to construct a sample path of any DES modeled as a stochastic timed automaton.

2.2.1. Sample Path Construction (SPC) procedure.

Step 1: Given the current state x_k and next event times $\tau_{i,k}$ for all feasible events $i \in \Gamma(x_k)$, determine the next event time:

$$t_{k+1} = \min_{i \in \Gamma(x_k)} \{\tau_{i,k}\} \tag{2.1}$$

Step 2: Determine the triggering event:

$$e_{k+1} = \arg \min_{i \in \Gamma(x_k)} \{\tau_{i,k}\} \tag{2.2}$$

Step 3: Determine the next state:

$$x_{k+1} = f(x_k, e_{k+1}) \tag{2.3}$$

Step 4: Update the next event times for all events $i \in \Gamma(x_{k+1})$:

$$\tau_{i,k+1} = \begin{cases} \tau_{i,k} & \text{if } i \neq e_{k+1} \text{ and } i \in \Gamma(x_k) \\ t_{k+1} + v_i(s_{i,k}+1) & \text{if } i = e_{k+1} \text{ or } i \notin \Gamma(x_k) \end{cases} \tag{2.4}$$

Step 5: Update the event scores:

$$s_{i,k+1} = \begin{cases} s_{i,k}+1 & \text{if } i = e_{k+1} \\ s_{i,k} & \text{otherwise} \end{cases} \tag{2.5}$$

Step 6: Increment k and continue from *Step* 1.

For some given state x_0, this procedure is initialized by setting $s_{i,0} = 0$ for all $i \in E$ and $\tau_{i,0} = v_i(1)$ for all $i \in \Gamma(x_0)$. For those familiar with DES simulation, the SPC procedure above is nothing more than the standard event scheduling simulation scheme. Some readers may have noticed that the formalism we have used here to define the GSMP is slightly different than the usual one (e.g., see [2, 12, 14]), but it will prove more useful for our purposes as we will subsequently explain.

2.3. Concurrent Sample Path Construction.

A considerable amount of work is currently being directed towards solving the sample path constructability problem. For DES in which all event processes are Markovian (memoryless), the

Standard Clock method [29] and Augmented System Analysis [5] provide two very efficient solutions. *Concurrent simulation* [3,4]is a general-purpose approach for DES; while it is not as efficient as the above two, it is applicable to DES with arbitrary event lifetime distributions. Next we will review the principles of concurrent simulation and then present an explicit algorithm that implements the idea.

The starting point is to consider a given DES operating under a specific parameter value θ. Assume that all events and their occurrence times are observable. Now recall that associated with every event $i \in E$ is a real-valued clock sequence $\mathbf{v}_i = \{v_i(1), v_i(2), \ldots\}$, where, $v_i(j)$ is the j-th lifetime of event i. Then, define

$$V_i(n) = \{v_i(1),\ldots,v_i(s_{i,n})\}, \quad i \in E \tag{2.6}$$

to be the sequence of observed lifetimes of event i after n total events have been observed.

The objective is to use the observed event lifetime sequences (2.6) and the SPC procedure to construct the hypothetical sample path that would result if the same DES were operating under some different parameter value $\hat{\theta} \neq \theta$. Let $k \leq n$ be the total number of events in this hypothetical sample path that we are constructing. Denote the corresponding event lifetime sequences in the constructed sample path by

$$\hat{V}_i(k) = \left\{v_i(1),\ldots,v_i\left(\hat{s}_{i,k}\right)\right\}, \quad i \in E \tag{2.7}$$

where $\hat{s}_{i,k}$ is the corresponding score of event i in the constructed sample path. Next define

$$\tilde{V}_i(n,k) = \left\{v_i\left(\hat{s}_{i,k}+1\right),\ldots,v_i\left(s_{i,n}\right)\right\}, \quad i \in E \tag{2.8}$$

to be sequences of those observed event lifetimes which have not yet been used in the construction of the hypothetical sample path; that is, these sequences contain all event lifetimes which are in $V_i(n)$ but not in $\hat{V}_i(k)$. We associate with $\tilde{V}_i(n,k)$ a set

$$A(n,k) = \left\{i : i \in E, s_{i,n} > \hat{s}_{i,k}\right\} \tag{2.9}$$

consisting of the subset of events i for which $\tilde{V}_i(n,k)$ contains at least one element, i.e., there is at least one observed lifetime available that has not yet been used in the constructed sample path. This set $A(n,k)$ is referred to as the *available event set* because it contains the set of events whose lifetimes are available to be used to construct the hypothetical sample path after n observed events. Lastly, we define one more set as follows. Let $\hat{x}_k$ denote the state after k events on the constructed sample path, and let $\hat{e}_k$ be the triggering event at the $(k-1)$-th state visited on this sample path. Then, define

$$M(n,k) = \Gamma(\hat{x}_k) - \left(\Gamma(\hat{x}_{k-1}) - \{\hat{e}_k\}\right) \tag{2.10}$$

That is, $M(n,k)$ contains all those events that are feasible in state $\hat{x}_k$ that were not feasible in state $\hat{x}_{k-1}$. Note that the triggering event $\hat{e}_k$ is also in this set if it happens that $\hat{e}_k \in \Gamma(\hat{x}_k)$. Intuitively, $M(n,k)$ consists of all those events whose occurrence times are missing from the perspective of the constructed sample path when it enters the state $\hat{x}_k$: The occurrence times for events in $\Gamma(\hat{x}_{k-1})$ are already known and remain available to be used in the sample path construction if they are still feasible; those events not feasible in $\hat{x}_{k-1}$ which have become feasible in the state $\hat{x}_k$, on the other hand, are *missing* as far as their occurrence times are concerned. We shall refer to $M(n,k)$ as the *missing event set* after n observed events.

In is clear from equations (2.1)-(2.2) of the SPC procedure in the last section, that in order to continue sample path construction from state $\hat{x}_k$ in the hypothetical sample path, we must have lifetimes (equivalently, occurrence times) for all events in the feasible event set $\Gamma(\hat{x})$. A key result from [4] is a necessary and sufficient condition for sample path construction: When event e_{n+1} is observed in the actual DES, construction of the hypothetical sample path can continue if and only if

$$M(n+1,k) \subseteq A(n+1,k) \tag{2.11}$$

otherwise, construction of the hypothetical sample path is "suspended" in state $\hat{x}_k$ until some future observed event causes (2.11) to be satisfied. Thus, with every observed event, condition (2.11) is checked: If it is satisfied, the SPC procedure is invoked to update the state of the constructed sample path; otherwise, construction is suspended at $\hat{x}_k$ until some future observed event causes (2.11) to be satisfied. Note that with every observed event the set $A(n,k)$ is updated and possibly enlarged, while the set $M(n,k)$ remains fixed, since it depends only on $\hat{x}_k$.

An explicit algorithm for constructing concurrent sample paths under parameter values $\hat{\theta} \neq \theta$ based on observed data from a sample path under θ is given in [4], where a detailed discussion of the conditions for which the approach is applicable may also be found. Briefly, the conditions for meaningful concurrent sample path construction are the following: (a) We assume that θ does not affect the event lifetime distributions in the DES, but only the state transition structure. If that is not the case, the algorithm described next requires certain modifications that we will not dwell on here. (b) Changes in θ should not introduce new types of events into the event set E. (c) The state transition structure of the observed system is assumed irreducible. More generally, there should be no state transition causing an event to become permanently disabled (since this implies that (2.11) may never be satisfied).

The algorithm in [4] is referred to as the *Time Warping Algorithm* (TWA). We reproduce it here with minor changes to suit our purposes. In the algorithm, the operators $+$ and $-$ are applied

to both scalars and sequences. When applied to scalars, they denote the usual addition and subtraction operations. When applied to sequences: + indicates the addition of an element to the end of the sequence; – indicates removal of the first element of the sequence.

2.3.1. Time Warping Algorithm (TWA).

1. *INITIALIZE*:

Given $x_0, \hat{x}_0$.

Set $n = k = 0$, $t_n = \hat{t}_k = 0$; for all $i \in E$ set $s_{i,n} = \hat{s}_{i,0} = 0$; for all $i \in \Gamma(x_0)$ set $\tau_{i,0} = v_i(1)$.

Set $M(0,0) = \Gamma(\hat{x}_0)$, $A(0,0) = \varnothing$

2. *WHEN EVENT e_n IS OBSERVED*:

   2.1. Use the SPC procedure to determine t_{k+1}, e_{k+1}, x_{k+1}, and $\tau_{i,k+1}$, $s_{i,k+1}$ for all i

   2.2. Add the observed lifetime of e_{n+1} to $\tilde{V}_i(n+1,k)$:

$$\tilde{V}_i(n+1,k) = \begin{cases} \tilde{V}_i(n,k) + v_i(s_{i,n}+1) & \text{if } i = e_{n+1} \\ \tilde{V}_i(n,k) & \text{otherwise} \end{cases}$$

   2.3. Update available event set: $A(n+1,k) = A(n,k) \cup \{e_{n+1}\}$

   2.4. Update missing event set: $M(n+1,k) = M(n,k)$

   2.5. If $M(n+1,k) \subseteq A(n+1,k)$, then go to 3.

        Else, set $n \leftarrow n+1$ and repeat from 2

3. *TIME WARPING OPERATION*:

   3.1. Obtain $v_i(\hat{s}_{i,k}+1)$ from $\tilde{V}_i(n+1,k)$ for all missing events $i \in M(n+1,k)$ and use the SPC procedure to determine $\hat{t}_{k+1}$, $\hat{e}_{k+1}$, $\hat{x}_{k+1}$, and $\hat{\tau}_{i,k+1}$, $\hat{s}_{i,k+1}$ for all i.

   3.2. Discard all used event lifetimes:

$$\tilde{V}_i(n+1,k+1) = \tilde{V}_i(n+1,k) - v_i(\hat{s}_{i,k}+1) \text{ for all } i \in M(n+1,k)$$

   3.3. Update available event set:

$$A(n+1,k+1) = A(n+1,k) - \{i : i \in M(n+1,k), \hat{s}_{i,k+1} = s_{i,n+1}\}$$

   3.4. Update missing event set:

$$M(n+1,k+1) = \Gamma(\hat{x}_{k+1}) - (\Gamma(\hat{x}_k) - \{\hat{e}_{k+1}\})$$

   3.5. If $M(n+1,k+1) \subseteq A(n+1,k+1)$ then set $k \leftarrow k+1$ and go to 3.1.

        Else, set $k \leftarrow k+1$ and $n \leftarrow n+1$ and go to 2.

Note that the time warping operation (i.e., steps (3.1)-(3.5)) may result in several state updates in the constructed sample path in response to a single observed event in the actual DES: As long as the sample path construction condition (2.11) is satisfied, construction of the hypothetical sample

path will proceed; otherwise, the constructed sample path's clock is stopped, while the observed system's clock keeps moving ahead. When the missing lifetimes become available and (2.11) is satisfied, the constructed sample path "instantaneously" processes as many events as possible causing its clock to "warp" forward. This process of moving backward in time to revisit suspended sample paths and then forward in time by one or more events lends itself to the term *time warping* [4].

It should be clear that by a simple modification to the TWA, any number of hypothetical sample paths can be concurrently constructed: instead of a single sample path under a parameter value $\hat{\theta}$, we can have many sample paths each indexed by $c = 1,2,...$ and each operating under a different parameter value θ_c. Computationally, the requirements of the TWA are minimal: adding/subtracting elements to sequences, simple arithmetic, and checking condition (2.11). It is usually the memory requirements that limit the number of concurrent sample paths that can be constructed, since the event lifetimes $\bar{V}_i^c(n,k)$ need to be stored for each constructed sample path c. The advantage of simultaneously constructing many sample paths, however, lies in the fact that from the full state history generated for each constructed sample path, it is possible to evaluate and compare any desired performance measure of interest. In this way the TWA can be used to solve the sample path constructability problem and the optimization problem (**P**).

2.4. The Speedup Factor.

If one is to compare the performance of the *Time Warping Algorithm* to the "brute force simulation", then the following *speedup factor* provides a convenient measure. Suppose that the sample path constructed through concurrent simulation were instead generated by a separate simulation and let T_N be the time it takes (in CPU time units) to generate N events. Also suppose that when a typical simulation is executed with *TWA* resulting in the construction of $K \leq N$ events, the total time is given by $T_N^0 + \tau_K$ where T_N^0 is the time to simulate the observed system without *TWA*. Clearly, by simulating such an observed path without and then with *TWA* allows us to easily obtain τ_K values in practice. We then define the *speedup factor* due to *TWA* as

$$S = \frac{T_N / N}{\tau_K / K} \tag{2.12}$$

Thus, if a separate simulation, in addition to the one for the observed sample path, were to be used to generate a sample path under a new value of the parameter of interest, the computation time per event is T_N / N. If, instead, we use concurrent simulation, no such separate simulation is necessary, but the additional time per event imposed by the approach is τ_K / K where $K \leq N$ in general. The ratio of the two is the speedup factor, which must obviously exceed 1 if concurrent

simulation is to be beneficial. Before presenting the simulation results that demonstrate the benefits of "concurrent simulation" we are going to obtain an upper bound of the speedup factor.

2.4.1. Speedup Upper Bound.

In many simulators, random number generation is a time consuming process. In this respect, *concurrent simulation* has a potential advantage over brute force simulation because it is able to cut down on the number of random numbers required by reusing the ones that have been observed in the nominal sample path. It is natural then to raise the question: "what is the potential speedup of *concurrent simulation*?" In this section we would like to determine an upper bound on the speedup that can be achieved through the use of concurrent simulation.

First, note that the simulation real time (i.e. actual CPU time) consists of two components.

- The time used to generate random numbers

- The time used to update the state, event list etc. (i.e. everything else other than generating random numbers)

Therefore we can write:

$$T_N = \alpha T_N + (1 - \alpha) T_N \tag{2.13}$$

where:

T_N: is the total CPU time to simulate N events in the nominal sample path.

α: is the percentage of time used for generating random variates ($0 \leq \alpha \leq 1$)

When we use *concurrent simulation,* we just save all lifetimes observed in the nominal sample path and then reuse them to construct the perturbed sample path. In this case, the total CPU time will be:

$$T'_K = (1 - \alpha) T'_K + rw_K$$

where rw is the time needed to write and read the necessary random numbers to and from the computer memory and also check that (2.11) is satisfied (i.e. rw constitutes the overhead of the concurrent simulation algorithm). Since the constructed sample path depends on the lifetimes observed in the nominal sample path, it is necessary that $K \leq N$ and therefore under the best case scenario all lifetimes that have been saved are used (i.e. $K=N$). Furthermore, given N, then the ratio T'_K / K decreases as K increases. This is true because most of the extra work (i.e. checking

$$-13-$$

(2.11) and saving the event lifetimes) which is used in the constructed sample path is done in the nominal sample path and it is not a function of K, therefore increasing K does not increase the overhead of the algorithm, however, it increases the amount of the overhead that is actually utilized, so the overhead is distributed over more events which implies that the time per event (i.e. T_K' / K) is a decreasing function of K and it takes its minimum value when $K=N$.

Now we are in a position to evaluate the *speedup*

$$S = \frac{T_N / N}{T_K' / K} \leq \frac{T_N}{T_N'} = \frac{T_N}{(1-\alpha)T_N + rw_N} \tag{2.14}$$

where we used the fact that $(1-\alpha)T_N = (1-\alpha)T_N'$ since excluding the random variate generation process the two sample paths are otherwise identical. Finally, define β to be the ratio of the time taken to write/read random variates to the memory over the time taken to generate the random variates, that is

$$\beta = \frac{rw_N}{\alpha T_N}$$

Substituting this in the speedup equation (2.14) we get:

$$S \leq \frac{T_N}{(1-\alpha)T_N + \alpha\beta T_N} = \frac{1}{1-(1-\beta)\alpha} \tag{2.15}$$

This implies that the maximum achievable speedup is always a function of the model we want to simulate through α, where we need to point out that α also depends on the actual implementation of the simulator. Furthermore, the speedup is also a function of the actual computer we are running the simulation on through β.

Figure 2.2 shows some plots of the *speedup* as a function of α for various values of β, where it can be seen that concurrent simulation is most beneficial when used for DES that require a large number of random variates and also for systems where the random variates required are obtained from complex distributions. Furthermore, note that for systems where $\alpha > 0.5$ it is possible to obtain speedup factors greater than 2 which implies that we reduce the actual time required to construct a sample path to less than half the time it would have taken if we had used brute force simulation.
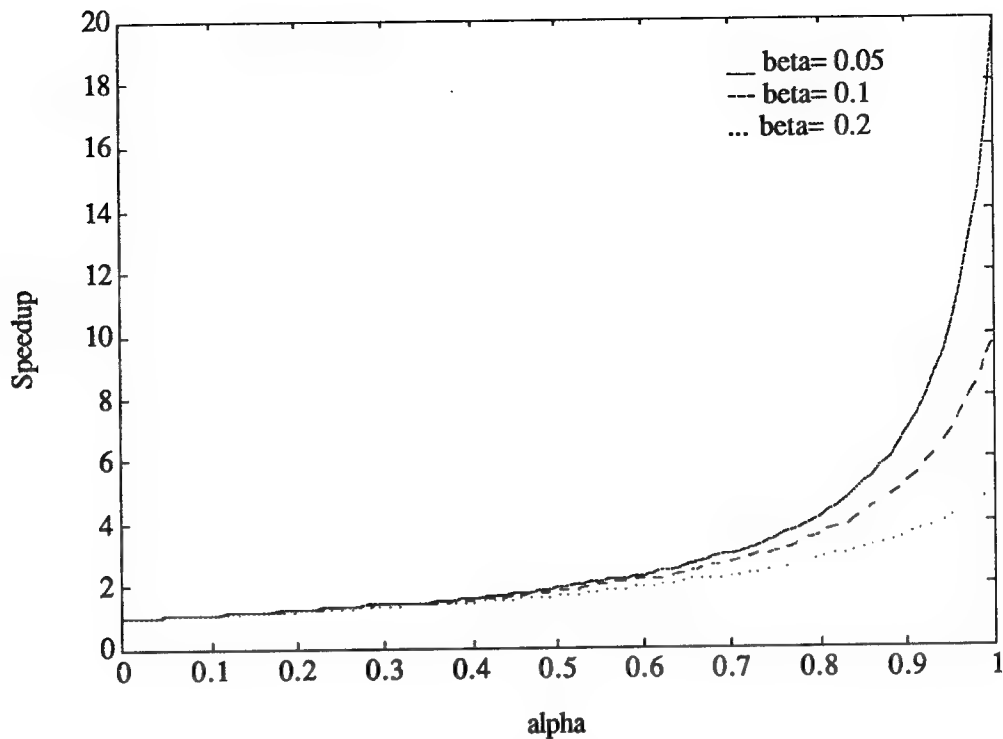
Figure 2.2. Speedup as a function of α.

2.5. Simulation Results.

Simulation can be used to readily verify that the *Time Warping Algorithm* generates sample paths identical to those generated by a separate simulation run with the same input. In this section, we shall focus on studying the speedup factors, as defined in (2.12), obtained for a variety of DES. We briefly describe below the specific DES considered with the speedup factors obtained.

System 1: *M/M/1/K* with multiple classes of customers.

This is a single-server queueing system serving various classes of customers on a First-In-First-Out (FIFO) basis. Systems with 2 to 11 classes of jobs were implemented and each class has exponential service and interarrival times. The performance measure considered is the blocking probability as a function of the buffer size K. Several values of arrival and service rates were used so as to achieve different server utilizations, as shown in **Table 2.1**. In addition, experiments included a system where one of the classes had a very low arrival rate (100 times slower) in order to investigate the behavior of *TWA* when a constructed sample path may be suspended for a long time while waiting for a lifetime it is missing.

System 2: *G/G/1/K* with multiple classes of customers.

This is the same as **System 1** with two classes of customers, but the service and interarrival times are now obtained from an Erlang distribution of the same order so that the server utilization is 0.67. As seen in **Table 2.1,** the speedup factor increases with the order of the Erlang distribution; this is expected, since Erlang random variate generation is more complex than the exponential one, and therefore α is increased.

System	Speedup Factor	Comments
1	2.44	Utilization 0.25
	2.20	Utilization 0.5
	2.44	Utilization 0.5+rare event
	2.18	Utilization 0.75
2	3.64	Erlang order 2
	7.73	Erlang order 5
	16.48	Erlang order 10
	27.00	Erlang order 20

Table 2.1. *Speedup factors for Systems 1-2.*

System 3: Two queues in series.

This is a serial network of two *M/M/1/K* queues where there are only two classes of jobs. Both servers operate under a FIFO scheduling policy. The arrival process is Poisson and the two servers are exponential. The performance measure of this system is the throughput as a function of the buffer allocation, i.e. (K_1, K_2) subject to $K_1 + K_2 = K$ where K_1, K_2 are the number of buffers allocated to each server. **Table 2.2** shows some typical speedup results obtained for different parameter settings under "manufacturing blocking"(i.e. jobs that find the second queue full wait in the first server for the next available queueing slot). The arrival rate of class i is λ_i and its service rate at the jth server is μ_{ij} $(i,j=1,2)$.

λ_1	μ_{11}	μ_{21}	λ_2	μ_{12}	μ_{22}	Speedup
1.0	4.0	4.0	1.0	4.0	4.0	2.63
1.0	3.0	3.0	1.0	3.0	4.0	2.58
1.0	4.0	4.0	1.0	4.0	3.0	2.63
1.0	3.0	3.0	1.0	3.0	3.0	2.25

Table 2.2. *Speedup factors for System 3.*

System 4: Three queues in series.

This is the same as **System 3** except we extend it to three queues. Two cases are considered: "manufacturing" blocking (as in system 3) and "communication" blocking where jobs that find the second queue full are lost. Typical results are shown in **Table 2.3**. One observation is that comparing **Tables 2.2** and **2.3**, the increase in size of the system from two queues to three had minimal effect on the speedup factors observed.

System	λ_1	μ_{11}	μ_{21}	μ_{31}	λ_2	μ_{12}	μ_{22}	μ_{32}	Speedup
manufacturing	1.0	5.0	7.0	12.0	1.0	5.0	7.0	12.0	2.70
blocking	1.0	3.0	3.0	3.0	1.0	5.0	7.0	9.0	2.67
	1.0	2.0	6.0	3.0	2.0	5.0	4.0	3.0	2.62
	3.0	20.0	4.0	4.0	3.0	20.0	4.0	4.0	2.41
communication	1.0	5.0	10.0	15.0	1.0	5.0	10.0	15.0	2.66
blocking	1.0	3.0	3.0	3.0	1.0	5.0	7.0	9.0	2.49
	1.0	2.0	5.0	3.0	3.0	5.0	7.0	5.0	2.52
	2.0	3.0	10.0	4.0	5.0	7.0	10.0	9.0	2.50

Table 2.3. *Speedup for System 4.*

System 5: Two queues in parallel with a single bulk-service server.

A single server is servicing two classes of customers using Round Robin scheduling policy. Both customer arrival processes are Poisson and each class is serviced in batches with exponential service times. Furthermore, there is a deterministic time delay every time the server switches from one class to the other, denoted by $SW_{1\rightarrow2}$ and $SW_{2\rightarrow1}$. The performance measure is the average system delay as a function of the batch size of each class of customers. Some representative results are shown in **Table 2.4**.

λ_1	μ_1	λ_2	μ_2	$SW_{2\rightarrow1}$	$SW_{2\rightarrow1}$	Speedup
1.0	2.0	1.0	3.0	0.2	0.4	2.51
1.0	2.5	1.0	5.0	1.0	0.5	1.91
1.0	2.0	1.0	2.0	0.5	0.0	1.74
1.0	2.0	1.0	2.0	0.0	0.0	2.11

**Table 2.4.** *Speedup for System 5.*

2.6. Conclusions and future work.

The concurrent simulation approach we have presented is intended to solve the sample path constructability problem described in Section 2.1 without imposing any restrictions on the event processes in the DES as in earlier work, however, it is subject to the standard trade off between generality and efficiency. The approach leads to the specific "time warping" algorithm (TWA) detailed in Section 2.3.1, which was analyzed and compared to the brute force simulation technique. Finally, we obtained several simulation results that show the benefits of concurrent simulation

The *TWA* as presented in Section 2.3.1 is able to give estimates of the performance of the DES for all m sample paths concurrently, however, in many applications this is not necessary, and it is more important to complete all m simulations as fast as possible. A new approach which is still under study is to decouple the observed and constructed sample paths. More specifically, when we simulate the observed sample path we just save all the observed event lifetimes. Then, we perform m-1 additional brute force simulations but rather than generating the required lifetimes all over again we can use the ones that have been observed in the nominal sample path. In this way, we eliminate the checking of (2.11) and the subtraction operation from the set $\tilde{V}_i(.,.)$ at the expense of more memory.

3. USING NEURAL NETWORKS FOR METAMODELING.

For many applications involving simulation, the time required to run the simulation may be very long or it may be necessary to perform many simulation runs. As described in Section 1, metamodeling addresses both of these issues: If one can develop a metamodel that captures the functional input/output relationships embodied by the simulator, then it is possible to obtain the simulation data quickly and efficiently.
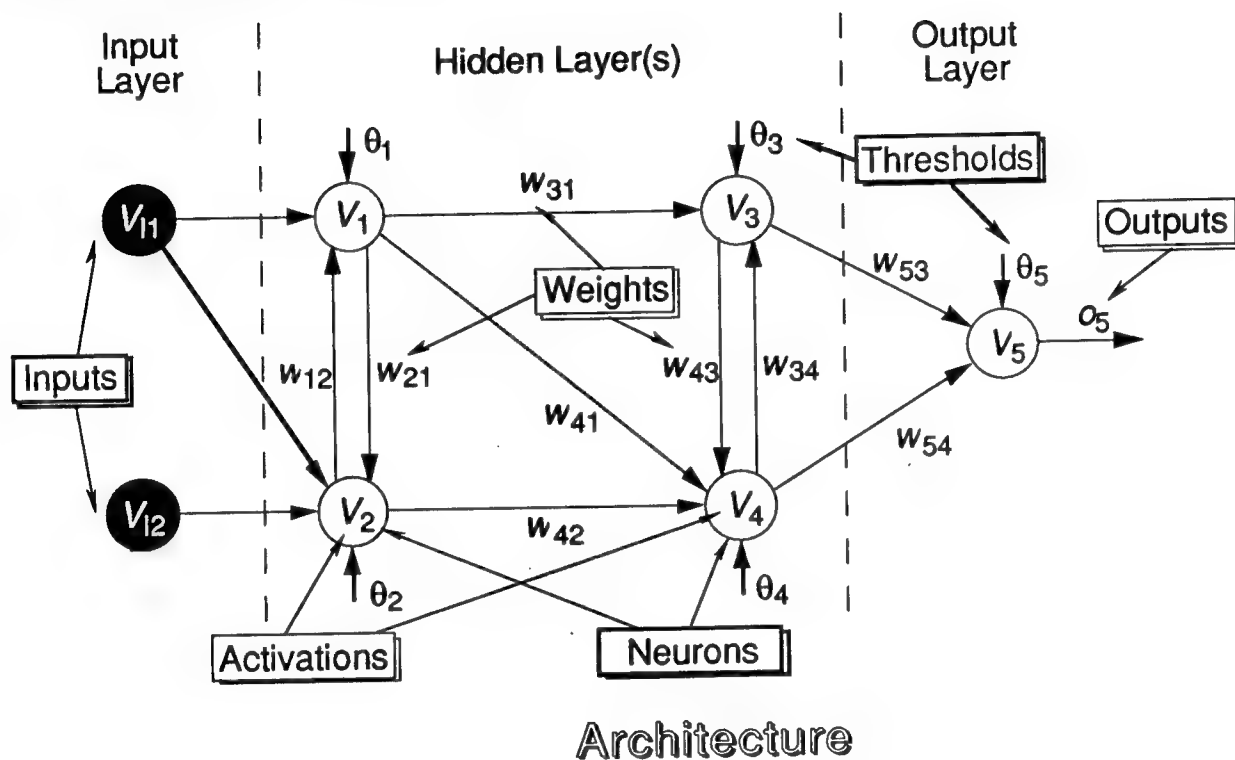
Figure 3.1: Architecture of a typical multilayer neural network

It is well known that multilayer neural networks are universal function approximators [8]. As such, they are often used for non-parametric modeling, and are well suited to the task of metamodeling. We begin with a brief description of the basic structure of a neural network. **Fig. 3.1** depicts the architecture of a typical multilayer neural network. In this particular picture there are four layers, including one input layer, one output layer and two intermediate layers called the "hidden layers". Each layer consists of many nonlinear devices called "neurons". The output of each neuron is called the "activation" and is denoted by V_i in **Fig. 3.1**. The basic operation of a neuron is defined by

$$Activation := g\left(\sum_{\substack{incoming \\ connections}} weights \times activations - threshold \right) \cdot$$

The *threshold* of the ith neuron is denoted by θ_i in the figure. The form of the function $g(\cdot)$ above is very important in the operation of a neural network. It is usually chosen to have the form of $g(x) = tanh(\beta x)$ and is called a "sigmoid function". It can be shown that such a structure with a proper "training" procedure could approximate almost all types of functions (e.g., see [11]).

As can be seen from **Fig. 3.1**, all the neurons are connected through links with different coefficients attached to each link. These coefficients are the "weights" in the above equation. The objective of training the network is to adjust the weights so that application of a set of inputs would produce the desired set of outputs. A typical training procedure can be described as follows:

1. Choose all the initial weights randomly
2. Select the training input-output pair from the training set
3. Calculate the output of the network
4. Calculate the error between the actual network output and the desired output
5. Adjust the weights of the network to minimize the error
6. Repeat steps 2-5 for each vector in the training set until the error for the entire set is acceptably low.

Step 5 is crucial and a special algorithm known as the "backpropagation algorithm" is often used. The basic idea of the backpropagation algorithm is actually quite simple and is an implementation of the usual "chain rule" in calculus in the special neural network structure. A description of such an algorithm would be too cumbersome to include here due to the many indices and sub-indices involved in the equations. For a representative description, however, see [11].

In summary, a neural network is a device (i.e., software) which conceptually consists of many "neurons" with "weights" attached to them. By adjusting the weights, a neural network is capable of generating response surfaces (i.e., outputs) of extreme generality. The idea of using "neurons" comes from the "stimulus-response" paradigm that our human brain neurons seem to conform to.

Based on the description above, the metamodeling procedure we have pursued is outlined next (see also **Fig. 3.2**). First, a large-scale simulation with possibly hundreds or thousands of inputs and hundreds or thousands of outputs is executed. The neural network is a device that we have separately designed (completely independently of the simulator) which is fed by the exact same inputs and outputs of the simulator; it does not, however, intrude in any way into it. While the simulator is running, the neural network observes the inputs and outputs and it "learns" from them. This is called "training" the network. One can visualize the neural network as an "entity" which is highly intelligent but has no knowledge of anything initially to apply its intelligence to. As it

observes the simulation unfold, however, it learns from the basic cause-effect (i.e., input-output) relationships it observes. In fact, all the neural network does is adjust its weights so as to emulate the behavior it observes as closely as possible.
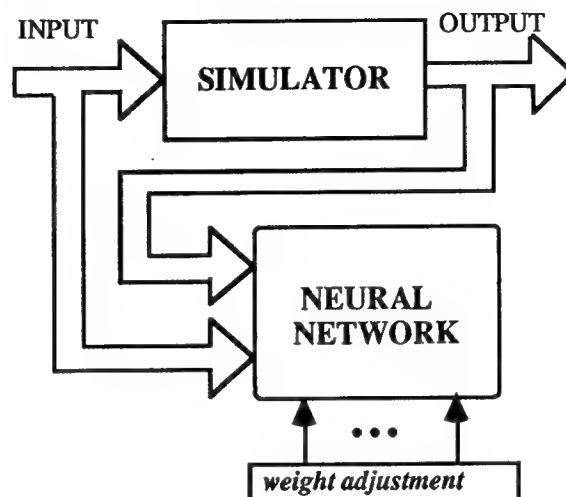
Figure 3.2: Neural network training through simulation to create a surrogate model

When the training is done, the simulator can be taken away. The neural network is now the surrogate model: we may give it some inputs (as if we were giving them to the simulator) and it immediately gives us an output (as the simulator would). So, we can think of it as a "function" which responds to any input by providing some output, except that there is no explicit mathematical expression or formula — just a device (a software routine) that acts as the model.

The main advantages of a neural network can be summarized as follows: (i) *Generality*: Neural network metamodeling has some indisputable advantages over the usual mathematical formulae one might use for metamodels. For example, when the system dimension or the inputs involved are in the hundreds or thousands, the mathematical formulae will simply become too complicated to use and lose the metamodeling feature. (ii) *Inherent Parallelism*: Neural networks can be viewed as generalized nonlinear regression tools with inherent parallelism (a very attractive feature). They are, therefore, ideal for modeling large-scale systems such as combat models. In fact, adjusting coefficients of some metamodel given in polynomial or rational function form may be completely infeasible if the dimension is indeed large. (iii) *Well-suited for model sensitivity analysis*: It is always possible to do model sensitivity analysis with a trained neural network, so long as the related factors have been arranged as the input to the neural network. Thus, there is an excellent opportunity here to combine this metamodeling approach with our concurrent/parallel techniques (which are also part of the proposed effort) to perform model sensitivity analysis on the neural network surrogate model. Preliminary experiments with simple discrete event systems have

demonstrated that this approach works fine and the neural network can indeed "learn" the behavior of the system reasonably quickly.

Clearly, to take full advantage of such an approach, one must design the neural network appropriately and develop efficient ways to accomplish the all-important training process. The key issues we have investigated in this project are: (a) Select an appropriate network architecture (e.g., the number of layers to use in a scheme such as that of **Fig. 3.1**), (b) For a manageable, but representative, example of the type of systems we are interested in studying, we need to identify a training set, (c) Study and analyze the backpropagation scheme (or related schemes we will develop) required to carry out the training of the neural network, (d) Test the accuracy of the resulting neural-network-based metamodel of the system we have selected.

Unfortunately, applying neural networks to real-world problems remains somewhat of an art. Without trial and error tuning, neural networks can take a long time to "learn" the intended task. In addition, it can be difficult to choose the correct network size. If the network is too small it will not be able to learn the task, and if it is too large it may "overfit" the data. In both cases, the resulting performance will be less than satisfactory. To address this major issue, we have used the *Cascade Correlation Neural Network* (CCNN) as a type of multilayer neural network that builds itself while it learns [10, 15, 24]. The CCNN starts small and makes itself larger during training which usually leads to faster learning and better performance. Section 3.1 is an overview of the CCNN. A high-level description of the CCNN training algorithm is given in Section 3.2. This is followed in Section 3.3 by a rigorous mathematical description of the training algorithm. Later, in Section 4, we use the CCNN to develop simulation metamodels.

3.1. Overview of the Cascade Correlation Neural Network (CCNN).

The Cascade Correlation Neural Network (CCNN) is a type of neural network that builds its own architecture as training progresses. It is based on the premise that the most significant difficulty with current learning algorithms for neural networks (e.g., backpropagation) is their slow rate of convergence. This is due, at least in part, to the fact that all of the weights are being adjusted at each stage of training. A further complication is that the network architecture is fixed throughout training.

A CCNN addresses both of these issues by dynamically adding hidden units to the architecture—but only the minimum number necessary to achieve a specified error tolerance for the training set. Furthermore, a two-step weight-training process ensures that only one layer of weights is being trained at any time. This allows the use of simpler training rules (e.g., the delta rule) than for multilayer training (e.g., backpropagation).

A CCNN consists of input units, hidden units, and output units. Input units are connected directly to output units with adjustable weighted connections. Connections from inputs to a hidden unit are trained when the hidden unit is added to the net and are then frozen. Connections from the hidden units to the output units are adjustable.

A CCNN starts with a minimal net, consisting only of the required input and output units (and a bias input that is always equal to 1). This net is trained until no further improvement is obtained; the error for each output unit is then computed (summed over all training patterns).

Next one hidden unit is added to the net in a two-step process. During the first step, a candidate unit is connected to each of the input units, but is not connected to the output units. The weights on the connections from the input units to the candidate unit are adjusted to maximize the *correlation* between the candidate's output and the residual error at the output units. The residual error is the difference between the target and the computed output, multiplied by the derivative of the output unit's activation function. When this training is completed, the weights are frozen and the candidate unit becomes a hidden unit in the net.

The second step in which the new unit is added to the net now commences. The new hidden unit is connected to the output units, the weights on the connections being adjustable. Now all connections to the output units are trained to minimize the mean squared error at the output units.

A second hidden unit is then added using the same process. However, this unit receives an input signal both from the input units and from the previous hidden units. All weights on these connections are adjusted and then frozen. The connections to the output units are then trained. The process of adding a new unit, training its weights from the input units and previously added hidden units, and then freezing the weights, followed by training all connections to the output units, is continued until error reaches an acceptable level or the maximum number of epochs is reached.

3.2. Building a CCNN.

A CCNN with 3 input units and 2 output units during the first stages of construction and learning is shown in **Fig's 3.3-3.8**. The bias input unit is shown by the symbol 1, it is a constant signal. The weights from the input units to the hidden units are denoted u. The weights from the previous hidden units to the new hidden units are denoted by t. The weights from the input units directly to the output units are denoted by w, and the weights from the hidden units to the output units are denoted by v.
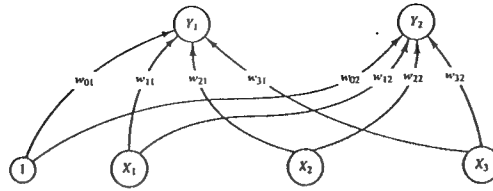
Figure 3.3. CCNN, Stage 0: No hidden units.
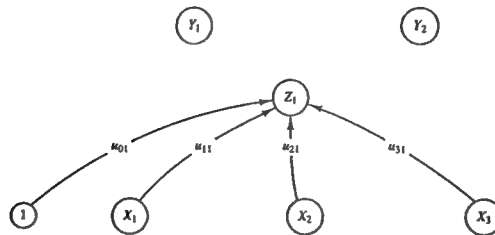
Figure 3.4. CCNN, Stage 1a: 1 candidate unit Z_1

Figure 3.4 shows the net at the first step of Stage 1. There is one candidate unit, Z_1, which is not connected to the output units. The weights shown are trained and then frozen.
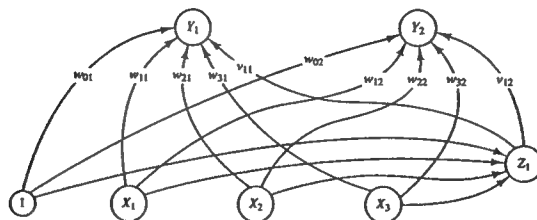
Figure 3.5. CCNN, Stage 1b: 1 hidden unit Z_1.

Figure 3.5 shows the second step of Stage 1. The hidden unit, Z_1, has been connected to output units. The weights shown in **Fig. 3.4** are now frozen. Weights to the output units are trained, and the error is computed for each output unit.
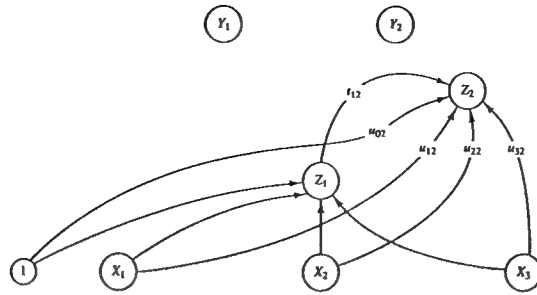
Figure 3.6. CCNN, Stage 2a: new candidate unit , Z_2.(showing weights being adjusted).

Figure 3.6 shows Stage 2. Here a new candidate unit, Z_2, takes signals from the input units and the previous hidden unit Z_1. Z_2 is not connected to output units during this stage. The weights shown are trained and then frozen. Weights on connections from X's to Z_1 are also frozen.

Figure 3.7. CCNN, Stage 2b: Hidden unit Z_2 connected to output units.

In **Figure 3.7**, the new hidden unit, Z_2, has been connected to output units. The weights shown in Figure 3.4 are now frozen. Weights to output units are trained, and the error is computed.
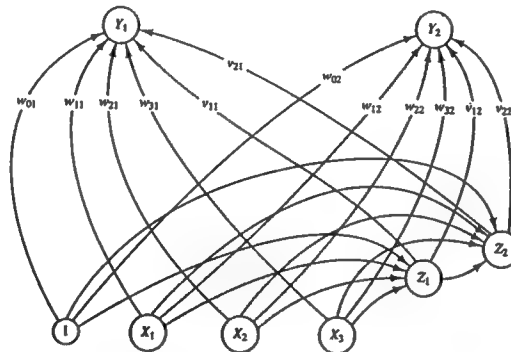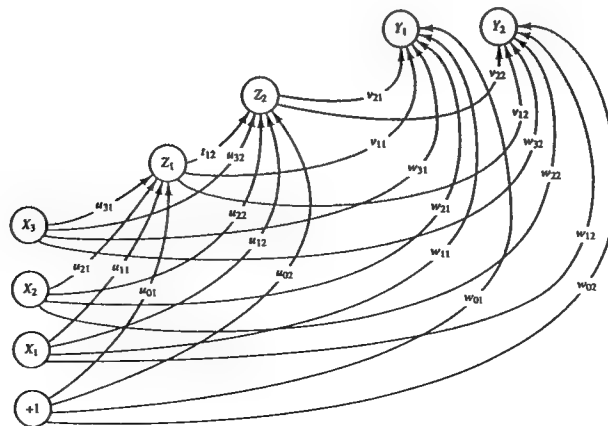

Figure 3.8. CCNN with 2 hidden units.

Figure 3.8 shows the same net as in **Fig's 3.3-3.7** with all weights shown. Weights u and t are frozen, and weights w and v are retrained at each stage.

3.3. The CCNN Training Algorithm.

The training process for CCNN involves both adjusting the weights and modifying the architecture of the net. First, some notation:

n	dimension of input vector (number of input units)
m	dimension of output vector (number of output units)
P	total number of training patterns
X_i	input units, $i=1,..., n$
Y_j	output units, $j=1,..., m$
$x(p)$	training input vector, $p=1,..., P$
$t(p)$	target vector for input vector $x(p)$
$y(p)$	target vector for input vector $x(p)$
$E_j(p)$	residual error for output unit Y_j for pattern p:

$$E_j(p) = (y_j(p) - t_j(p))\dot{y}_j(p)$$

E_av_j average residual error for output unit Y_j:

$$E\_av_j = \frac{1}{P}\sum_{p=1}^{P} E_j(p)$$

z(p)	computed activation of candidate unit for input vector x(p)
z_av	average activation, over all patterns p=1,...,P, of candidate unit:

$$z\_av = \frac{1}{P}\sum_{p=1}^{P} z(p)$$

In the algorithm that follows, the weights on all new connections are initialized to small random values. Biases are included on all hidden units and on all output units. This is usually indicated by including an additional input unit whose activation is always 1.

The *correlation* is defined as

$$C = \sum_{j=1}^{m}\left|\sum_{p=1}^{P}(z(p) - z\_av)(E_j(p) - E\_av_j)\right|$$

z_av   average activation, over all patterns p=1,...,P, of candidate unit:

$$z\_av = \frac{1}{P}\sum_{p=1}^{P} z(p)$$

In the algorithm that follows, the weights on all new connections are initialized to small random values. Biases are included on all hidden units and on all output units. This is usually indicated by including an additional input unit whose activation is always 1.

The *correlation* is defined as

$$C = \sum_{j=1}^{m}\left|\sum_{p=1}^{P}(z(p)-z\_av)(E_j(p)-E\_av_j)\right|$$

This is actually the covariance between the output of the candidate, z, and the "residual output error". The residual output error is the product of the true error and the derivative of the activation function for the output unit.

In a manner similar to that for the derivation of the backpropagation we find that

$$\frac{\partial C}{\partial u_i} = \sum_{j=1}^{m}\sigma_j\sum_{p=1}^{P}\dot{z}(p)x_i(p)(E_j(p)-E\_av_j)$$

where σ_j is the sign of

$$\sum_{p=1}^{P}(z(p)-z\_av)(E_j(p)-E\_av_j)$$

$\dot{z}$ is the derivative of the activation function for the candidate unit, and x_i is the input signal received by the candidate unit from input unit X_i.

Training can use any standard method, such as the delta rule for training a single-layer net. The weights are adjusted to minimize the error in Steps 1 and 3 of the algorithm and to maximize C in Steps 2 and 4. The activation functions may be linear, sigmoid, Gaussian, or any other differentiable functions, depending on the application.

The algorithm for cascade correlation is as follows:

Step 1.   Start with required input and output units.

Train the net until error reaches a minimum:

If error is acceptably small, stop;

If not, compute $E_j(p)$ for each training pattern p, E_av_j, and proceed to Step 2.

Step 5.   While stopping condition is false, do Steps 6 and 7.

(Add another hidden unit.)

    Step 6.   A candidate unit Z is connected to each input unit and each previously added hidden unit.

                Train these weights to maximize C

                When the weights stop changing, they are frozen.

    Step 7.   Train all weights v to the output units

                If acceptable error or maximum number of units has been reached, stop.

                Else continue.

Note that only one layer of weights is being trained during the unit's candidate phase. The weights from the input units to the previously added hidden units have already been frozen; only the weights to the candidate from the input units and other hidden units are being trained. when this phase of learning stops, those weights are frozen permanently.

Variations of this technique include several candidate units at Step 3 or Step 6 and then choosing the best candidate to add to the net after training. This is especially beneficial in a parallel computing environment, where the training can be done simultaneously. Starting with different initial random weights for each candidate reduces the risk of the candidates getting stuck during training and being added to the net with its weights frozen at undesired values.

4. TACTICAL ELECTRONIC RECONNAISSANCE SIMULATOR.

In this section we present our metamodeling studies using the *Tactical Electronic Reconnaissance Simulator* (TERSM). TERSM has been extensively studied and has been used by previous researchers to develop and evaluate polynomial metamodels [7,31,32]. As such, it provides an excellent testbed for our initial metamodeling studies. In Section 4.1 we give an overview to describe what TERSM is and how it works. The polynomial metamodeling approach is explained in Section 4.2. Sections 4.3 and 4.4 compare the CCNN metamodeling approach to the polynomial metamodeling approach for two different TERSM problems. The first TERSM problem (Section 4.3) is from the literature, and allows us to compare the CCNN metamodeling approach to existing research. The second TERSM problem (Section 4.4) is one of our own creation, and was designed specifically to try to expose weakness in our CCNN metamodeling approach. We close in Section 4.5 with a summary of the lessons learned.

4.1. TERSM Overview.

TERSM was developed by the RAND corporation for the United States Air Force [28]. It is a very complex simulator that models the flight of a reconnaissance aircraft carrying a bearing angle measuring sensor over a radar field. As shown in **Fig. 4.1**, the aircraft flies with a fixed heading at a constant altitude and a constant velocity over a battlefield which contains many ground-based radar sites. As the aircraft flies, the sensor records bearing angle measurements of the emitters it detects and builds a list containing the circular error probable (CEP) for each one. The CEP is a disk of such size that the probability that the emitter is inside the disk is 50%. The simulator was originally developed to compare competing sensors when making purchase decisions.
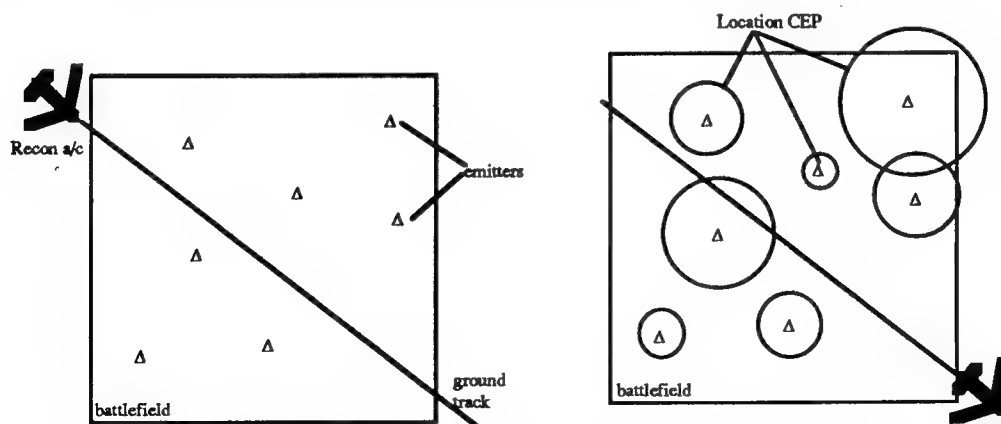

Figure 4.1. TERSM, start of mission (left), end of mission (right).

As illustrated in **Fig. 4.2**, TERSM works in a manner similar to the familiar police scanner that one can buy in an electronics store. Like a police scanner, the sensor scans a set of frequency channels in some preprogrammed sequence. It dwells on a channel for a short period of time waiting for an emitter to transmit on that channel. If an emitter is transmitting, a series of tests are performed to check if the sensor can detect it. The emitter must be within the line of sight of the sensor and not blocked by terrain or the curvature of the earth. The emitter must be within the field of view of the sensor's antenna pattern. The signal received by the sensor must be strong enough to detect, but not so strong that it exceeds the sensor's dynamic range. If all these tests pass, the sensor records the detection data. After a short dwell time, the sensor switches to another channel. During the brief time period required to switch from one channel to the next, the sensor processes all the detections made on the previous channel to extract bearing angle measurements. The channel capacity is a measure of the speed of the processing and indicates the number of detections which can be processed during the brief switching time. Complete details about how TERSM operates can be found in [28].
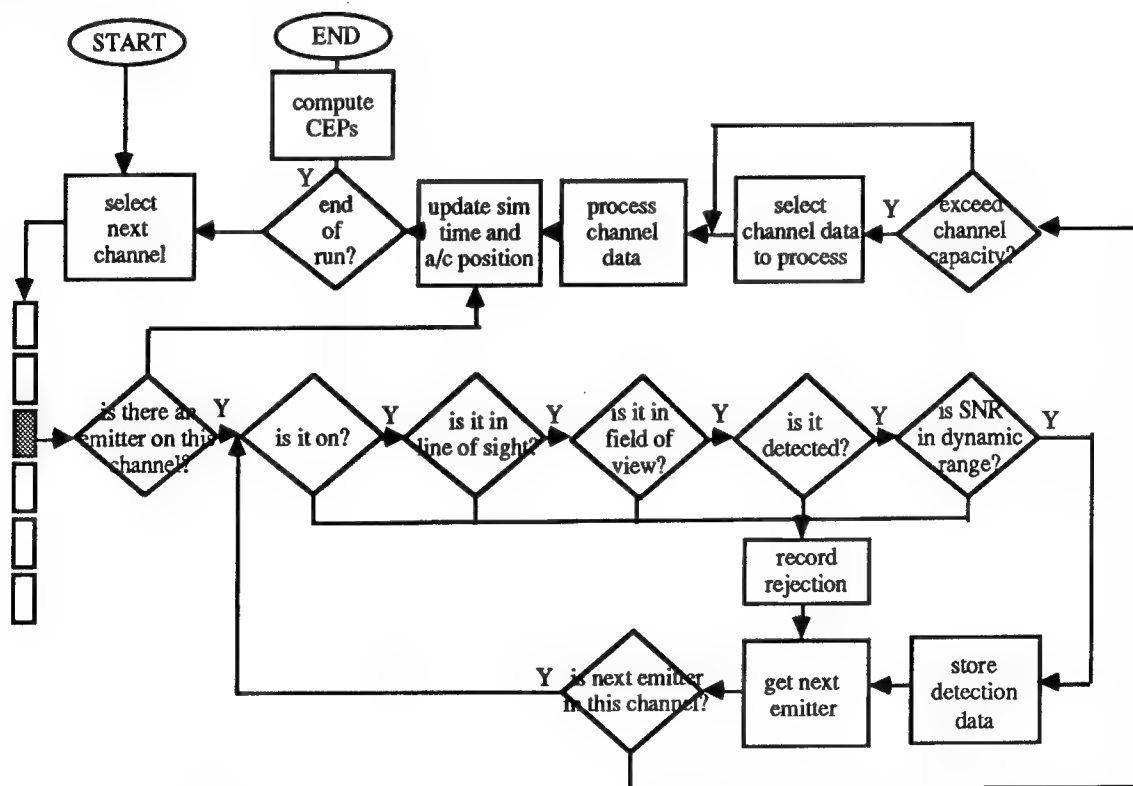
Figure 4.2. Flowchart illustrating how TERSM works.

The TERSM model has many inputs and is capable of generating a large volume of output data. As illustrated in **Fig. 4.3**, the inputs concern the aircraft flight data, the sensor data, and data for each emitter. **Table 4.1** gives a brief list of some of the inputs. For complete details see [28].
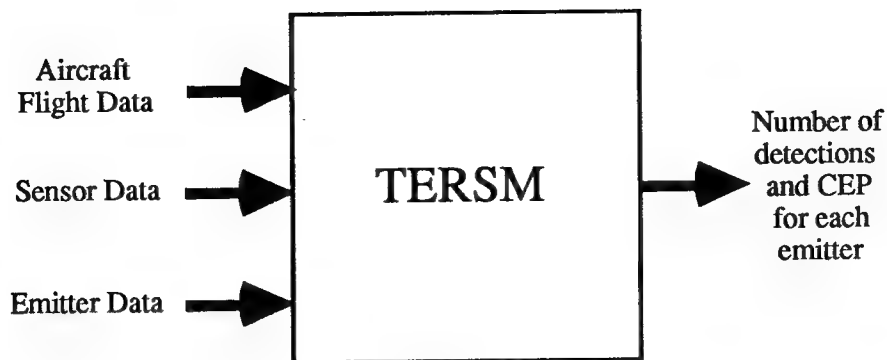
Figure 4.3. TERSM I/O summary.

Table 4.1. Abbreviated list of TERSM inputs.		
Aircraft Data	Sensor Data	Emitter Data

– initial position
– heading
– altitude
– velocity
	– frequency limits
	– channel bandwidth
	– dwell and scan times
	– detection threshold
	– elevation/azimuth
	  coverage angle
	– dynamic range
	– bearing error σ
	– channel capacity
	– channel scanning
	  sequence
		– type
		– location
		– frequency
		– pulse repetition
		  frequency
		– pulse width
		– mainlobe/sidelobe gain
		– power
		– azimuth/elevation info

For our studies, we were only interested in the aircraft and sensor inputs. All of our experiments use the same default emitter configuration used in [7,31,32]. This default emitter configuration has 2359 emitters contained within the boundaries of the rectangular region shown in **Fig 4.4**.

Figure 4.4. All 2359 emitters in the default emitter field are contained in the gray region.

4.2. Polynomial Metamodels.

For our initial feasibility studies we compared Cascade Correlation Neural Network (CCNN) metamodels to polynomial metamodels. In this section we give a brief overview of polynomial metamodeling.

The functional mapping represented by a polynomial is determined by its coefficients and its order. An example of a 2-input, 1-output, second order polynomial is given below.

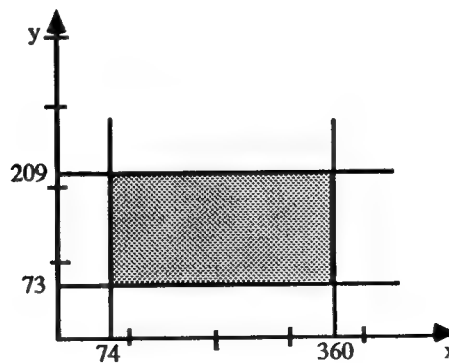$$\hat{y} = b_0 + b_1 x_1 + b_2 x_2 + b_3 x_1 x_2 + b_4 x_1^2 + b_5 x_2^2$$

The *least squares* approach gives a way to determine the coefficients which minimize the mean squared error between the output of the polynomial and the output of the function to be approximated. To explain the approach, suppose we wish to use the second order polynomial above to approximate some arbitrary two input, single output function,

$$y = f(x_1, x_2)$$

Since there are 6 coefficients to determine we must conduct *at least* 6 experiments. Suppose we have conducted $N \geq 6$ such experiments. Let us index each experiment by $n = 0,...N$. Designate the input points for the n-th experiment as $x_1(n)$ and $x_2(n)$ and the output of the polynomial for the n-th experiment as $\hat{y}(n)$. Then we can write,

$$\begin{bmatrix} \hat{y}(1) \\ \hat{y}(2) \\ \vdots \\ \hat{y}(N) \end{bmatrix} = \begin{bmatrix} 1 & x_1(1) & x_2(1) & x_1(1)x_2(1) & x_1^2(1) & x_2^2(1) \\ 1 & x_1(2) & x_2(2) & x_1(2)x_2(2) & x_1^2(2) & x_2^2(2) \\ & & & \vdots & & \\ 1 & x_1(N) & x_2(N) & x_1(N)x_2(N) & x_1^2(N) & x_2^2(N) \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_5 \end{bmatrix}$$

Using matrix notation, we can rewrite the above equation as,

$$\hat{y} = Xb$$

Here $\hat{y}$ is the vector of outputs, b is the vector of coefficients, and the matrix X is called the regressor matrix. Each row of the regressor matrix X is called a regressor vector. Let us designate the output of the function we are trying to approximate for each pair of inputs as

$$y = \begin{bmatrix} y(1) \\ y(2) \\ \vdots \\ y(N) \end{bmatrix} = \begin{bmatrix} f(x_1(1), x_2(1)) \\ f(x_1(2), x_2(2)) \\ \vdots \\ f(x_1(N), x_2(N)) \end{bmatrix}$$

The objective is to choose the coefficients b which minimize the mean squared error, i.e.,

$$\min_b \left[e^T e \right]$$

where,

$$e = y - \hat{y}$$

A necessary condition for a minimum is

$$\frac{\partial}{\partial b} \left[e^T e \right] = 0$$

Taking the indicated partial derivative yields

$$\frac{\partial}{\partial b} \left[e e^T \right] = \frac{\partial}{\partial b} \left[(y - Xb)^T (y - Xb) \right]$$

$$= \frac{\partial}{\partial b} \left[(y^T y - y^T Xb - b^T X^T y + b^T X^T Xb) \right]$$

$$= -2X^T y + 2X^T Xb$$

Solving for b gives

$$b = \left(X^T X \right)^{-1} X^T y \tag{4.1}$$

Some of the potential difficulties with the least squares approach should be immediately apparent. The difficulties are primarily centered around the inversion of the matrix $\left(X^T X \right)$. The size of this matrix is determined by the number of coefficients to be determined. The number of coefficients is determined by the number of inputs, and the order of the polynomial. If the number of coefficients is B then the size of the matrix we must invert is B by B. Since it can be difficult to invert large matrices, the least squares approach does not scale well to problems with many inputs and high order polynomials. We must also be very careful in choosing the N input points to ensure that the matrix is well conditioned and nonsingular.

As for all function approximation methods, the quality of the approximator depends on the particular set of N input points we use to obtain the coefficients. Since, we assume that we know very little about the function we are trying to approximate, selecting the best set of input points can be difficult. Methods for choosing the input data points fall under the general heading of *experimental design*. For first and second order polynomials there are well established experimental design methods for choosing the input points. For first order polynomials, the most

popular designs are the *orthogonal designs* [21,23], so called because they result in a diagonal $(X^T X)$. These designs are useful because they minimize the variance of the coefficients when there is noise in the output of the function being approximated. One way to obtain an orthogonal design is with a 2^k *factorial design* (where k is the number of inputs). To explain the method, suppose that the objective is to approximate an arbitrary function over a region of the input space defined by $L_i \le x_i \le U_i$ where L_i is the lower limit for the i-th input variable and U_i is the upper limit for the i-th input variable. For the method we need to collect 2^k I/O pairs. Each input point is chosen to be a corner of the input space. For example, to fit the curve in **Fig. 4.5**, we would choose the two input points, $x(1) = L$ and $x(2) = U$.
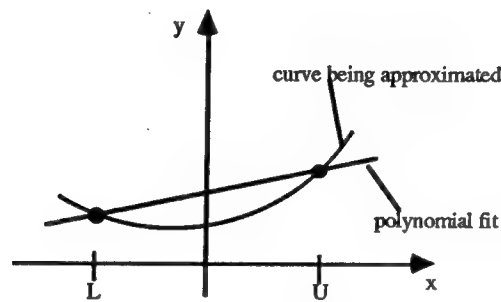
Figure 4.5. A 2^k factorial experiment for a first order polynomial curve fit.

When there is noise, it is sometimes useful to augment the input with several measurements at the midpoint $x_i = U_i - L_i / 2$. Usually, the inputs are *centered and scaled* so that the lower limit L_i maps to -1, the upper limit U_i maps to $+1$, and the midpoint maps to the origin. Thus, after the inputs are centered and scaled, the regressor matrix is a matrix of -1's, 0's, and 1's.

For second order polynomials, the most common design method is the *central composite design* (CCD) [21]. After the inputs are centered and scaled, a CCD consists of,

    1) 2^k factorial corner points (i.e., each x is either -1 or $+1$),

    2) $2k$ axial points (i.e., the points $(\pm\alpha,0,...,0)$, $(0,\pm\alpha,0,...0)$ etc.),

    3) n central points (i.e., the point $(0,0,...,0)$).

For a 3-input problem, these points are shown in **Fig. 4.6**. If we let $F = 2^k$, then choosing $\alpha = (F)^{\frac{1}{4}}$ gives a *rotatable* central composite design. **Table 4.2** from [21] indicates the number of central points needed to obtain a *uniform precision* design or an *orthogonal design*. As an example, an orthogonal second order CCD when the number of inputs $k = 4$ requires $2^k + 2k + n = 16 + 8 + 12 = 36$ test points. For higher order polynomials, design methods are not as well established, but one common approach is to use *layered* CCD's.
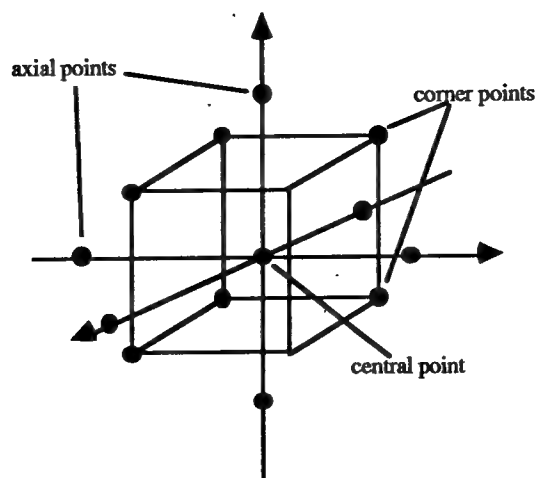
Figure 4.6. Collecting data for a second order polynomial using the central composite design.

Table 4.2. Number of central points needed for a uniform precision CCD and an orthogonal CCD.				
k input variables	2	3	4	5
n for uniform precision	5	6	7	10
n for orthogonal	8	9	12	17

4.3. Baseline TERSM Problem.

The baseline TERSM problem focused on the relationship between the number of emitters detected which have a CEP less than 5 nautical miles in radius and the following four inputs: the altitude of the aircraft (it flies at a constant altitude for the duration of its mission), the velocity of the aircraft (it flies at a constant velocity for the duration of the mission), the azimuth of the sensor (the angle of the sensor boresight relative to the aircraft heading), and the channel capacity of the sensor (the number of bearing angle measurements that can be processed during the channel switching time). This four input single output example (see **Fig. 4.7**) has been used in the literature to develop polynomial metamodels [7,31,32]. Since it is so well understood, this TERSM problem provides an excellent baseline for our feasibility studies.

In [7,31,32] polynomial metamodels were developed for the baseline TERSM problem. These polynomial metamodels were designed to capture the response surface over the range of inputs shown in **Table 4.3**.
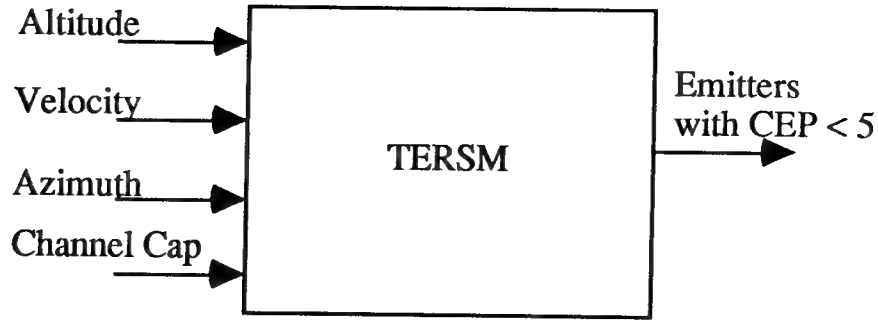
Figure 4.7. Baseline TERSM problem.

Table 4.3. Range of interest for each input variable.		
Input Variable	Lower Limit	Upper Limit
Altitude (feet)	5000	40,000
Velocity (knots)	186	1150
Azimuth (degrees)	60	150
Channel Capacity	4	30

4.3.1. Comparative Results.

Caughlin [7] compiled a table of 49 input/output pairs from which he obtained the following reduced order polynomial metamodel,

$$\sqrt{y} = 22.4331 - 0.0148x_1 - 2.7822x_2 + 0.1432x_3$$
$$+3.1432x_4 + 0.3653x_1x_3 + 1.2439x_1x_4 + 0.1483x_2x_3$$
$$+0.4430x_2x_4 + 0.2698x_3x_4 + 0.4369x_1x_2x_3 + 0.3286x_1x_2x_4$$
$$+0.0960x_1x_3x_4 - 0.2791x_2x_3x_4 - 0.8326x_1^2 + 0.7642x_3^2 \qquad (4.2)$$
$$-1.8413x_4^2 + 0.7577x_1^3 + 4.9038x_2^3 + 1.0924x_3^3 - 1.1907x_1^4$$
$$-4.8443x_2^4$$

In the equation above, the input x_1 is the altitude, x_2 is the velocity, x_3 is the azimuth angle, and x_4 is the channel capacity. All inputs are centered and scaled (so that the upper limit maps to 1, the lower limit to -1, and the middle value to 0). Using a version of the TERSM software given to us for this project, we were able to duplicate Caughlin's results. This served to validate our version of the TERSM simulation software.

Next, we randomly generated 238 sets of input points. Then we ran the TERSM simulator for each of the 238 input points and constructed a table of 238 input/output pairs. We used this table of 238 input/output pairs to train a CCNN metamodel.

Finally, we randomly generated an additional 165 sets of input points. We ran the TERSM simulator for each of the 165 input points and constructed a table of 165 input/output pairs. This table of 165 input/output pairs was used as a test set to compare the polynomial metamodel to the CCNN metamodel.

The performance measure we use is the *mean squared error* between the output of the metamodel (polynomial or CCNN) and the output of the TERSM simulator. For each input/output pair $i = 1,...,n$ in the data set, the mean squared error is computed as,

$$MSE = \frac{1}{n}\sum_{i=1}^{n} e_i^2 \qquad (4.3)$$

where e is the error or difference between the output of the metamodel and the TERSM output. **Table 4.4** below summarizes our results.

Table 4.4. Comparative performance between the polynomial and CCNN metamodels.			
	49 point set	238 point set	165 point set
polynomial	518.40	575.86	795.20
CCNN	497.75	353.25	417.50

In **Table 4.4**, the 49 point set is the set of input/output pairs that were used to obtain the polynomial coefficients. This is the *training set for the polynomial*. The 238 point set is the set of input/output data pairs that were used to train the CCNN. This is the *training set for the CCNN*. The 165 point set is independent of the other two (i.e., has no points in common with either of them) and represents an independent *testing set*. The purpose of the testing set is to see how well the metamodels generalize to data which they were not exposed to during training. Generalization involves both *extrapolation* to parts of the input space which are outside the range seen during training, and it involves *interpolation* to parts of the input space which lie "between" points seen during training.

As expected, the polynomial does well on its training set, and the CCNN does well on its training set also. The CCNN, however, does much better than the polynomial on the testing set. Since a metamodel is primarily going to be used to generalize, good performance on the testing set is the most important performance measure.

In comparing the two metamodeling approaches, we see that, constructing polynomial metamodels is somewhat of an art. There are various ways of deciding when certain terms are not significant, in which case their coefficients are set to zero. In equation (4.2), for example, the terms $x_1 x_2$, $x_1 x_2 x_3 x_4$, x_2^2, x_4^3, x_3^4, x_4^4 were determined to be insignificant and eliminated. In addition, for many polynomial curve fitting problems, it is advantageous to use a nonlinearity on the output. For the baseline TERSM problem, many nonlinearities were tried and the square root was found to give the best results [32].

Constructing CCNN metamodels is easier as it simply involves randomly selecting of a set of training points. Depending on the topology of the response surface, however, the number of required training points to get a good fit may be very large. For this problem, however, the response surface is relatively "smooth" and we obtained a good fit with only a small number of training points.

Figures 4.8-4.9 show how the size of the network and the number of training points affect the resulting mean squared error (MSE) of the CCNN. As can be seen, the MSE falls rapidly as the number of hidden units is increased, with best performance coming from a network with just over 10 hidden units. The CCNN training algorithm (see Section 3) automatically stopped adding hidden units when it reached 19. This shows that the CCNN can find the best network size automatically during training. It adds units as needed to fit the training data. Once it gets a reasonable fit, it stops adding units to avoid overfitting the data. We also see, that for this particular metamodeling problem, the MSE is essentially independent of the size of the training set. These two findings suggest that the baseline TERSM problem is not particularly challenging.

4.4. A "Tougher" TERSM Problem.

That a fourth-order polynomial gives a good metamodel for the baseline TERSM problem suggests that the baseline TERSM problem is not sufficiently challenging. For the baseline TERSM problem, the number of emitters detected changes in a more or less "continuous" way with changes in the input variables, leading to a relatively "smooth" response surface. For example, increasing the altitude allows the sensor to see further over the horizon and detect more emitters. Increasing the velocity results in less emitters being detected because the aircraft is over the emitter field for a shorter period of time. The azimuth was found to have little effect, and increasing the channel capacity results in a marginal increase in the number of emitters detected. Thus, the baseline TERSM problem has a response surface which is easy to fit with a low order polynomial.
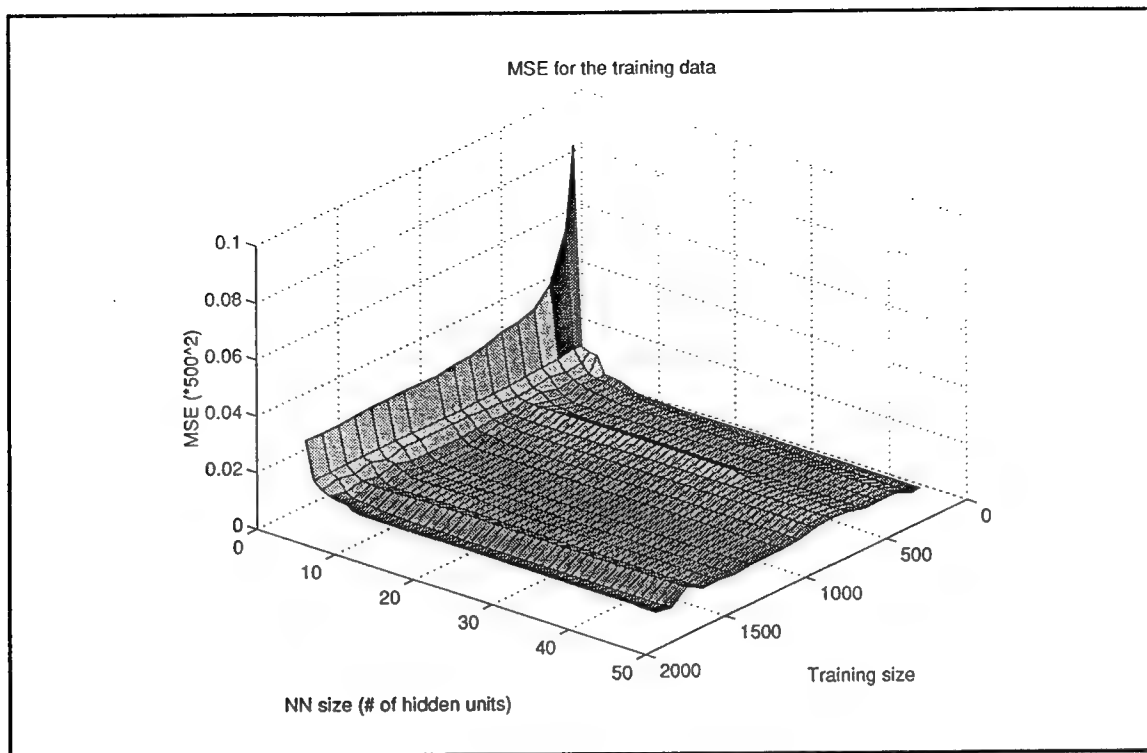
Figure 4.8. MSE for the training set.
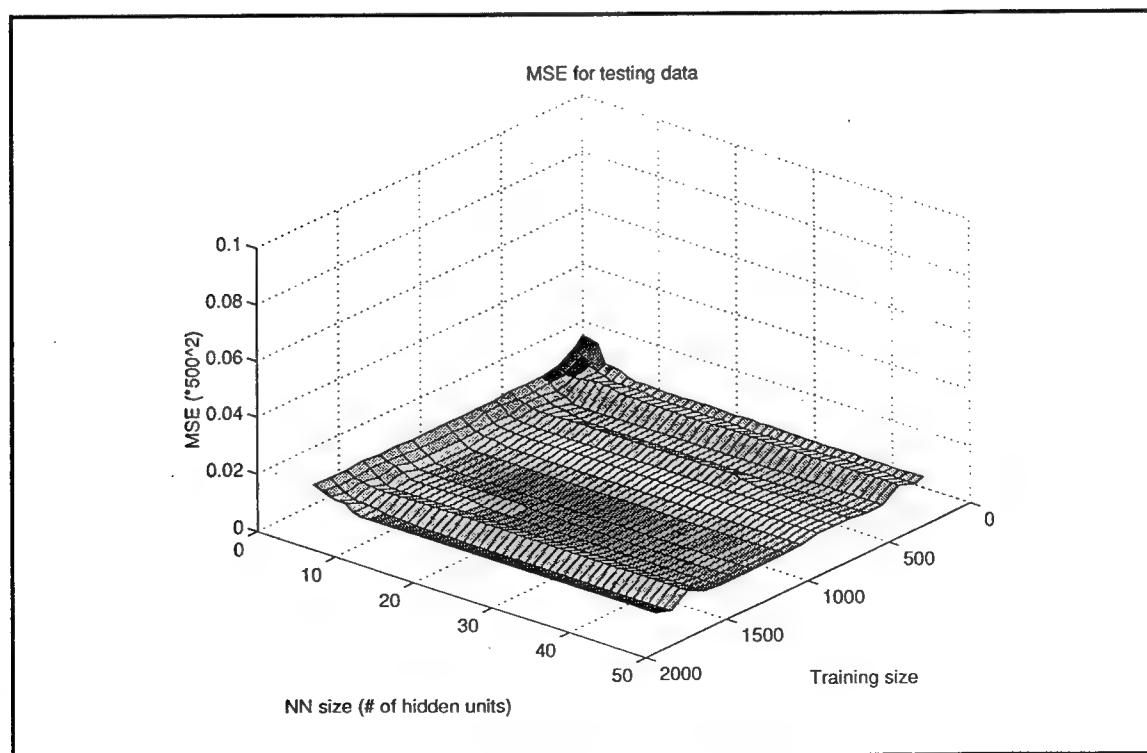
Figure 4.9. MSE for the testing set.

This prompted us to develop a "tougher" TERSM problem that has a "rougher" and more challenging response surface. For the tougher TERSM problem we looked at the functional relationship between the four inputs and single output shown in **Fig. 4.10**. As for the baseline TERSM problem, the output is the number of emitters detected which have a CEP less than 5 nautical miles. For the tougher TERSM problem the four inputs were: the initial aircraft coordinates Xo and Yo, the initial aircraft compass heading θ (the aircraft flies a constant heading), and the aircraft velocity V (the aircraft flies at a constant velocity).
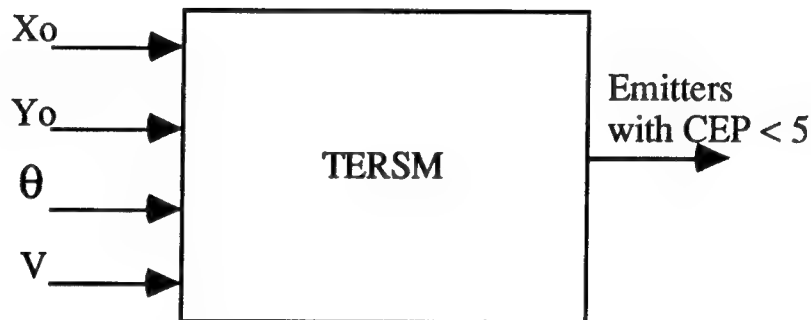
Figure 4.10. "Tougher" TERSM problem.
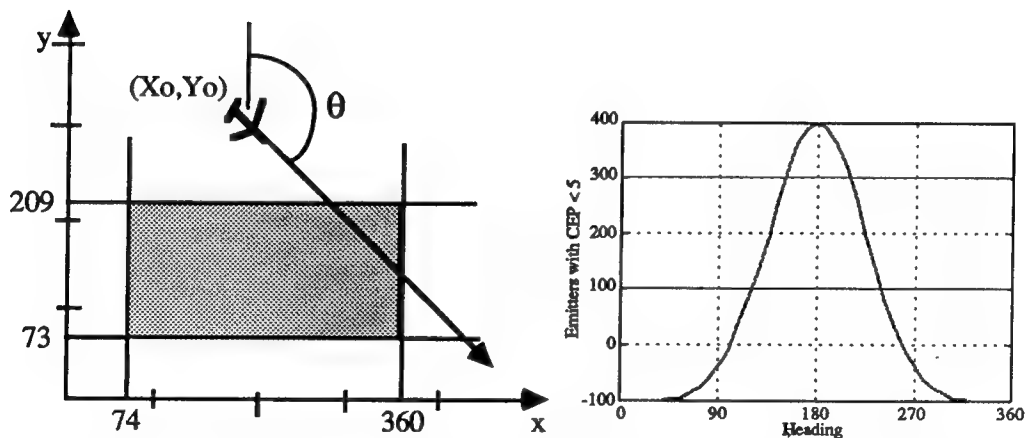
Figure 4.11. Sweeping heading from 0 to 360 degrees gives a "hump" in the response surface.

To get some insight as to why this TERSM problem is "tougher" consider **Fig. 4.11**. Imagine fixing the initial aircraft position and the aircraft velocity and varying just the aircraft heading. As we vary the heading, we will get a "hump" in the response surface. For a zero heading angle (due north) we are flying away from the emitter field and no emitters are detected. As the angle increases towards 180 degrees (due south) more and more emitters are detected as we fly over an increasingly larger portion of the emitter field. Then as the angle increases beyond 180 degrees, the number of emitters detected decreases. Thus, the tougher TERSM problem has a

response surface consisting of a collection of isolated "humps" where many emitters are detected, surrounded by large flat regions were no emitters are detected. Such a response surface should be quite difficult for a low order polynomial to capture. The CCNN, however, due to its universal function approximation capabilities, should be able to approximate the response surface. Nevertheless, such a response surface still poses substantial difficulties for the CCNN in terms of data collection. If the input points are not carefully chosen, it is possible to miss some the humps entirely.

For the tougher TERSM problem, our interest was to obtain a metamodel for the range of inputs shown in **Table 4.5**.

Table 4.5. Range of interest for each input variable.		
Input Variable	Lower Limit	Upper Limit
Xo	0	400
Yo	0	300
Heading (degrees)	0	359
Velocity (knots)	186	1150

As before, we performed simulation experiments to construct three data sets: a training set for the polynomial, a training set for the CCNN, and a testing set with which to compare the two.

4.4.1. Polynomial Results.

The polynomial training set was obtained using a layered CCD design that consisted of the 49 input/output pairs shown in the Appendix in **Table A.1**. As for the baseline TERSM problem, we found that using a square root nonlinearity on the output gave the best results.

$$
\begin{aligned}
\sqrt{y} = {}& 20.3885 - 4.8499x_1 + 7.7436x_2 - 2.6979x_3 \\
& -2.9343x_4 - 1.2951x_1x_2 + 3.6486x_1x_3 + 1.2144x_1x_4 \\
& +1.9655x_2x_3 - 2.3306x_2x_4 + 0.0265x_3x_4 - 1.0556x_1x_2x_3 \\
& -1.0158x_1x_2x_4 + 1.4222x_1x_3x_4 + 0.2683x_2x_3x_4 \\
& -1.0927x_1x_2x_3x_4 - 13.5657x_1^2 - 12.1073x_2^2 + 0.6690x_3^2 \\
& +3.1144x_4^2 + 6.4287x_1^3 - 9.1515x_2^3 + 3.3922x_3^3 + 5.2131x_4^3 \\
& +4.7740x_1^4 + 4.6920x_2^4 - 0.6565x_3^4 - 2.9759x_4^4
\end{aligned}
\tag{4.4}
$$

In equation (4.4), x_1 is the initial aircraft x-location, x_2 is the initial aircraft y-location, x_3 is the aircraft heading (0 degrees is due north), and x_4 is the aircraft velocity. All inputs are centered and scaled, and the output y is square root of the number of emitters located with a CEP less than 5

nautical miles. In this case there are 28 coefficients in the b vector, and each row of the regressor matrix X looks like,

$$\phi = [1, x, x_2, x_3, x_4, x_1 x_2, x_1 x_3, x_1 x_4, x_2 x_3, x_2 x_4, x_3 x_4, x_1 x_2 x_3, x_1 x_2 x_4,$$
$$x_1 x_3 x_4, x_2 x_3 x_4, x_1 x_2 x_3 x_4, x_1^2, x_2^2, x_3^2, x_4^2, x_1^3, x_2^3, x_3^3, x_4^3, x_1^4, x_2^2, x_3^2, x_4^4]$$

The ruggedness of the response surface resulted in the polynomial giving very poor performance. Even on the 49 point set, the one used to obtain the polynomial coefficients, the MSE was 37,279. That is, on the average, the output of the polynomial and the output of TERSM differed by 193 emitters, a huge difference when one considers that the number of emitters detected with a CEP < 5 nautical miles averaged only 243 and never exceeded 849 (see Appendix A, Table A.1).

4.4.2. CCNN Approach—First Try.

Initially, we had some difficulty getting good performance from the CCNN. We began by training the CCNN using 500 randomly selected data points, which resulted in a network with 50 hidden units (as opposed to 19 for the baseline TERSM problem). The increased size of the network is an indication that this is indeed a "tougher" problem. Even with such a large network, however, the MSE on the CCNN training set was quite large at 12,300.

To understand why the CCNN was not doing well, we generated plots to compare the output of TERSM to the output of the trained CCNN. In **Fig. 4.12** we fixed the velocity at 600 knots, and the heading at 100 degrees and plotted the TERSM output $y \sim F(x_0, y_0)$ as a function of the initial aircraft x and y position. Compare **Fig. 4.12** to **Fig. 4.13** which shows the output of the trained CCNN $\hat{y} \sim \hat{F}(x_0, y_0)$ for the same conditions. In **Fig. 4.14** we fixed the velocity at 600 knots and the initial aircraft y location at 150 and plotted the TERSM output $y \sim G(x_0, \theta)$ as a function of the initial aircraft x location and aircraft heading θ. Compare **Fig. 4.14** to **Fig. 4.15** which shows the output of the trained CCNN $\hat{y} \sim \hat{G}(x_0, \theta)$ for the same conditions.

As previously suggested, the response surface for the tougher TERSM problem is characterized by isolated humps surrounded by flat regions. With only 500 training points, we see that although the accuracy is poor, the CCNN is still able to get a rough "grasp" of the shape. Note that we could have used expert knowledge about the TERSM problem to identify where the humps in the response surface might be so that we could concentrate our data collection efforts there. However, we want to assume that such a domain expert is not available, as is often the case in practice (especially for very complex hierarchical systems).

4.4.3. New Issues.

In trying to understand why the CCNN did not perform well on the tougher TERSM problem, we asked ourselves the following questions.

1. *How large should the training data size be?*
In principle what is the optimal size? Can we use some kind of adaptive mechanism to find or approach this optimum? Can we segment the problem and develop separate models for different regions of the input space?

2. *How can we improve learning efficiency, and speed up the learning process?*
Is our CCNN architecture appropriate for the task? What if we introduce a non-linear activation function to the output unit? Is there an adaptive mechanism for adjusting the learning rate that we can use to speedup the training? Will starting from several different initial points (multi-starts) help training?

3. *How well can the cascade-correlation algorithm model nonlinearities?*
A more basic question is: How does a fixed architecture learning algorithm like multilayer neural network with backpropagation perform when doing the same job? Is there any basic limitation in terms of the ability of the CCNN to model certain types of nonlinearities?

Answers to the above questions may provide the major clues for improving the CCNN as a metamodeling tool. Some of these questions are addressed in the following subsections, others are not yet fully understood and will require further research.
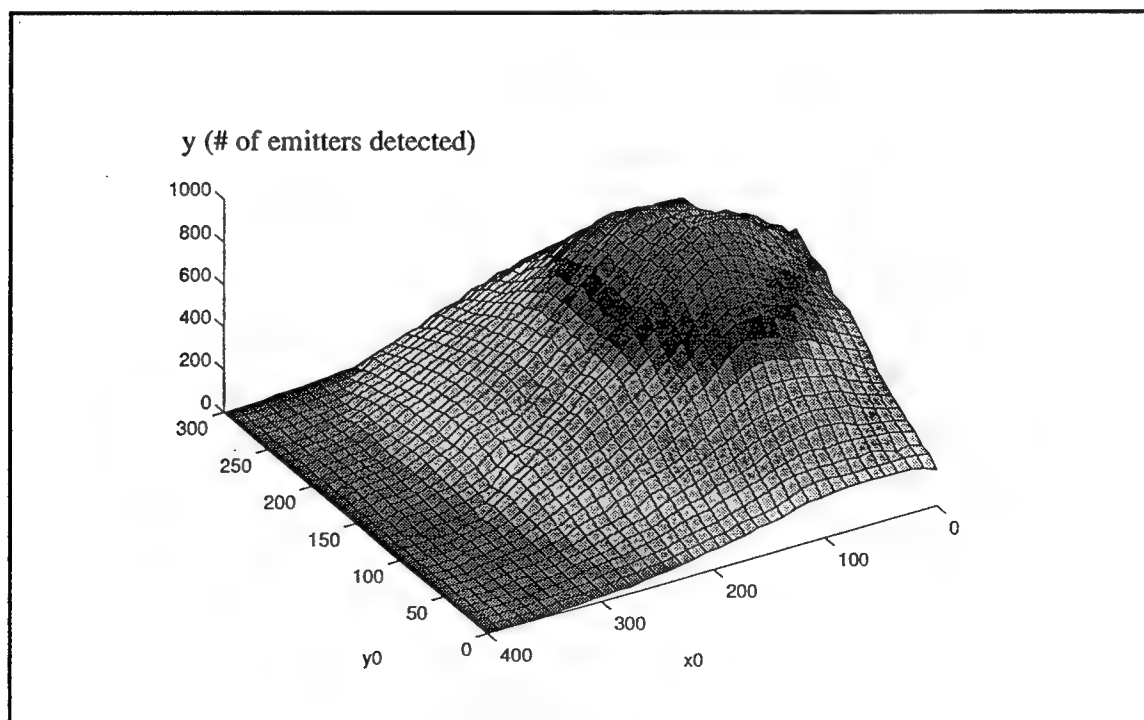

Figure 4.12. $y \sim F(x_0, y_0)$ velocity $= 600$, $\theta = 100$ (true TERSM output).
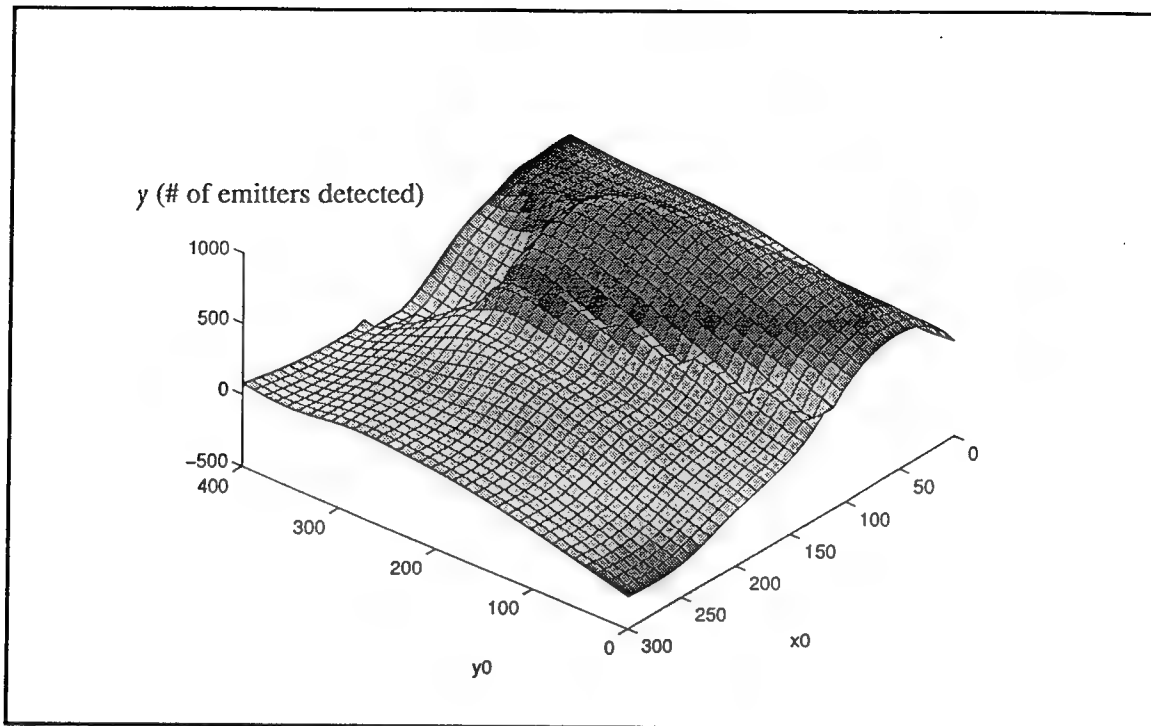
Figure 4.13. $\hat{y} \sim \hat{F}(x_0, y_0)$ velocity = 600, θ = 100 (CCNN output).
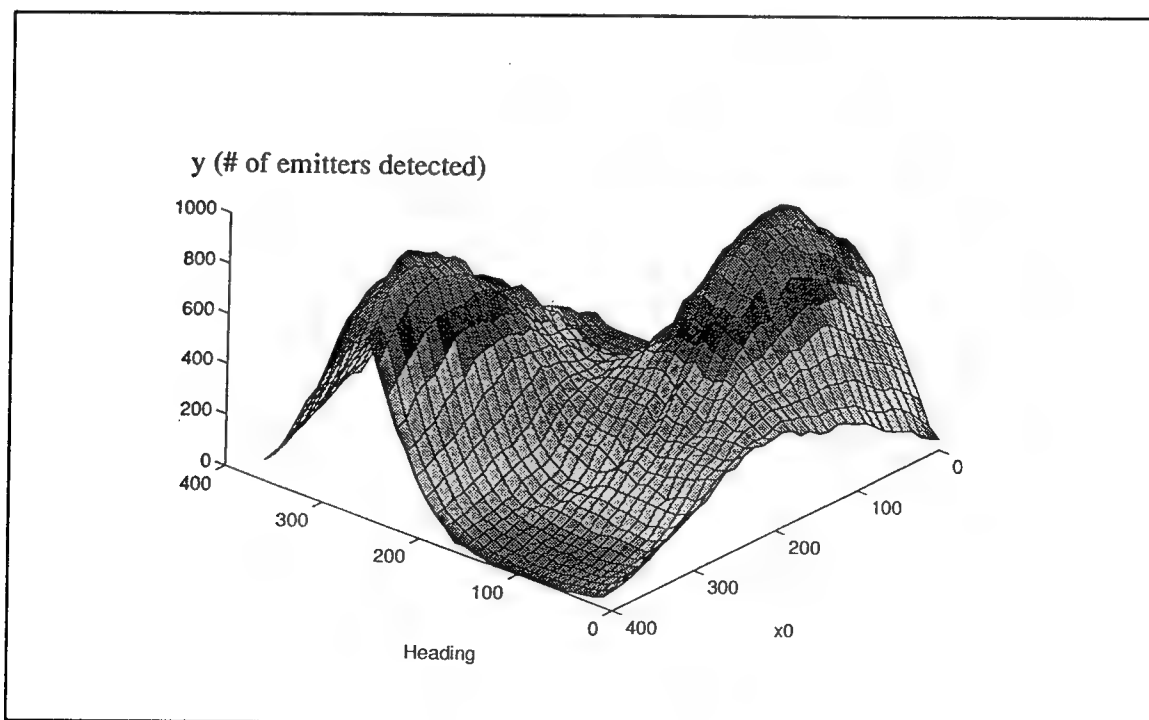
Figure 4.14. $y \sim G(x_0, \theta)$ velocity = 600, y_0 = 150 (true TERSM output).
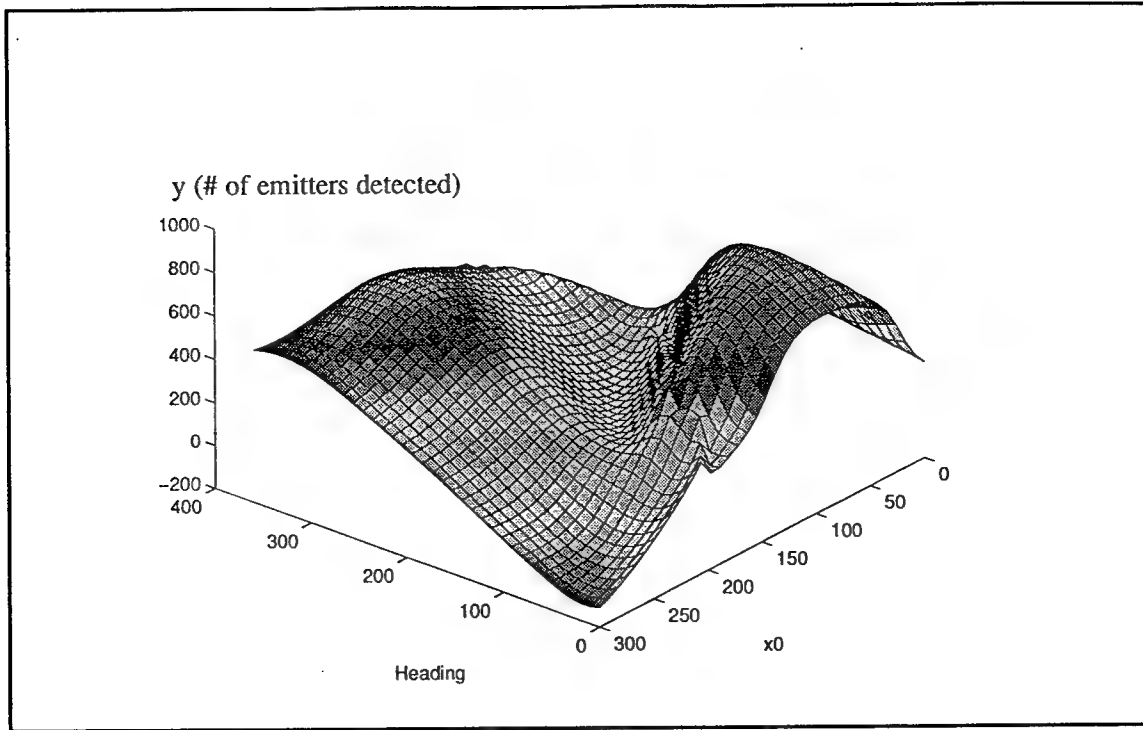
Figure 4.15. $\hat{y} \sim \hat{G}(x_0, \theta)$ velocity = 600, y_0 = 150 (CCNN output).

4.4.4. Study of CCNN Modeling Capability.

In this section we study the third question posed above. We begin by comparing the modeling capabilities of a standard multilayer neural network (MLNN) with the modeling capabilities of the CCNN. Recall, the standard MLNN has essentially the same structure as a CCNN except that its size is fixed and does not change during training. One might conjecture that since their structures are similar, their modeling capabilities should also be similar. However, because they use different training algorithms, this may not be the case. The CCNN training algorithm, for example, may put it at a disadvantage for certain types of modeling problems, specifically those involving isolated humps.

To see if the MLNN could perform better than the CCNN on the tougher TERSM problem, we built a MLNN with 1 input layer, 1 hidden layer with 100 hidden units, and 1 output layer. The MLNN was trained using the standard error-backpropagation algorithm with a fixed learning rate [26]. Our results showed that the MLNN did not outperform the CCNN. In fact, the CCNN learned much faster than the MLNN. We think that the slow convergence of the MLNN was mainly due to the choice of the learning rate, the initial values for the weights, and so on. The difficulty in selecting these parameters was one of the primary motivations for the development of the CCNN, which is much more automated and frees the practitioner from the trial and error chore of parameter selection.

-45-

After confirming that the MLNN does not offer advantages over the CCNN for the tougher TERSM problem, we wanted to see if maybe it is the structure of the response surface that is causing the difficulties. That is, does the CCNN training algorithm have difficulty learning curves which are characterized by flat regions with isolated humps. To see we designed a single variable test function consisting of a half-cycle of a sinusoid in the middle of a flat region. **Figure 4.16** shows the results of a CCNN trained with 200 randomly generated points. Although the test function looks relatively simple, modeling it required 33 hidden units in the CCNN. What makes the test function complex, and requires so many hidden units, is the presence of the "corners" or discontinuities at the junctions between the flat region and the hump. Such discontinuities will cause problems for any modeling scheme.

Next we tried a more difficult test function consisting of two humps. Again, the CCNN was trained using 200 randomly generated points. In this case, the CCNN grew to 67 hidden units, twice as large as when there was a single hump. The results are shown in **Fig. 4.17**. Notice that for this test function, the corners are much sharper than they were for the previous test function. In addition, there are more of them. Even so the CCNN captures all of the corners pretty well, except for the ones between the two humps.
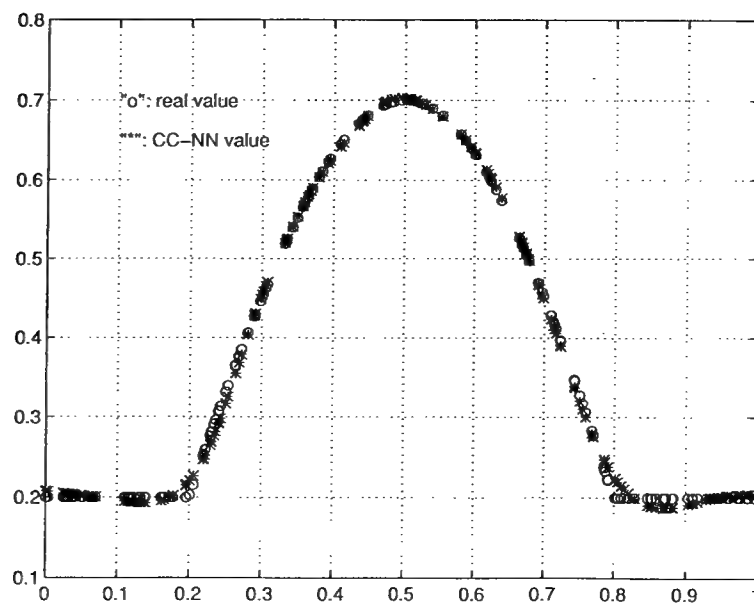
Figure 4.16. CCNN modeling of a curve with a single hump.
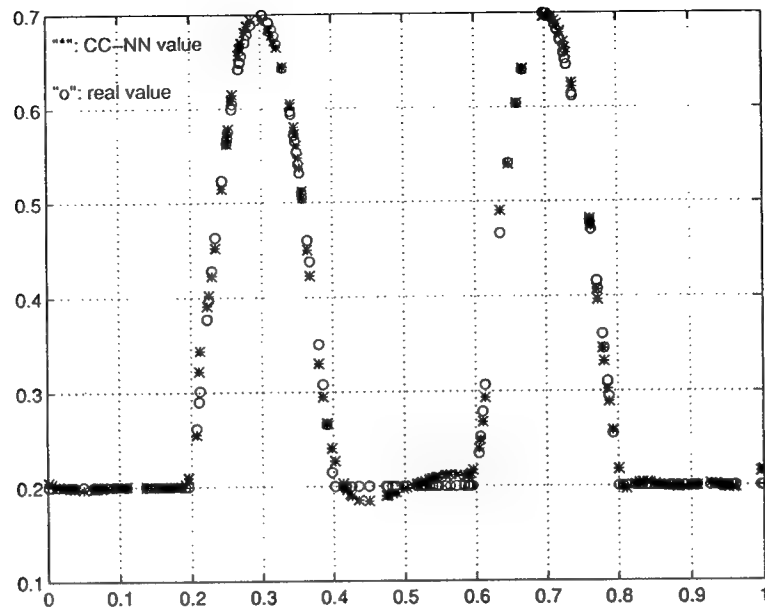
Figure 4.17. CCNN modeling of a two-hump curve.

In summary, we can conclude that the CCNN has no limitations in terms of its ability to learn surfaces of the sort that characterize the tougher TERSM problem. What we can conclude is that it is probably necessary to use more training points. For the test functions, the CCNN did not perform well, until we used 200 training points. For the tougher TERSM problem, however, we only used 500 training points. This is fewer than 5 points along each of the 4 input dimensions. For subsequent experiments, we increased the size of the training set to 14,641, which gives 11 points along each input dimension.

4.4.5. CCNN Algorithm Modifications.

In an attempt to improve the convergence rate and generalization capabilities of the CCNN, we tried some variations to the basic CCNN training algorithm.

1. Learning with momentum:

To improve training efficiency we added a "momentum" term to the weight updating rule. As is evident from equation (4.5), momentum works by adding a fractional portion of the previous weight change to the current weight change. This adds "inertia" to the weight updating process: If the previous weight change was large, the weights will tend to continue to move in the same direction. Adding a momentum term has been shown to improve the learning by "stabilizing" the

learning procedure [13]. Instead of getting caught in local minima in the weight space, the algorithm tends to leap right over them on its way toward the global minimum.

$$\Delta w_n = a \cdot \Delta w_{n-1} + \alpha \cdot \delta \cdot x \qquad (4.5)$$

2. Multi-starts:

Another modification we made is the introduction of multi-starts [27]. Namely, when adding a new hidden unit to the CCNN, we compete several candidate units each of which starts with different initial conditions and is independently trained (**Fig. 4.18**). After all of the candidates have been trained, the one with the maximum correlation index wins. We keep this winner as the new hidden unit and throw away the others. The use of multi-starts is intended to avoid the problem of getting stuck in a local minimum, a common problem for gradient-based training algorithms.

3. Sigmoidal activation function at output units:

Instead of using a linear activation function at the output, we tried using the same sigmoid nonlinearity that is used for the hidden units. The idea is that this may help the CCNN to deal with more complex shapes.
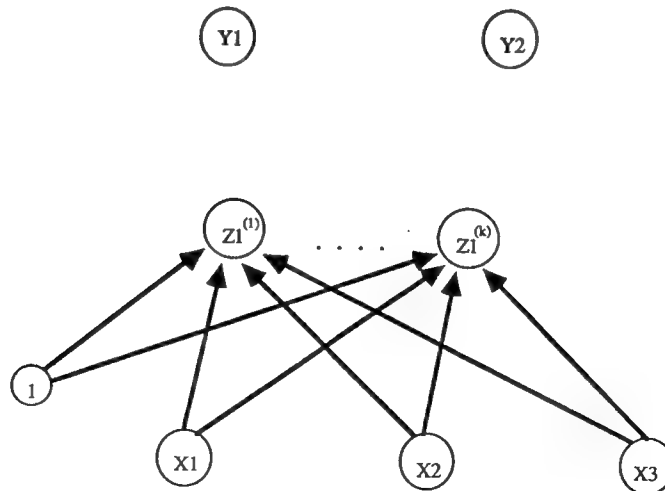
Figure 4.18. Multiple candidates competing to become hidden unit z1.

4. Learning rate annealing:

After learning has taken place for some time, it is usually the case that the weight updates become less and less dramatic, eventually settling to some final values. One way to encourage this behavior and increase the learning rate is to introduce learning rate "annealing"—let the learning rate slowly decrease, or "cool down" as learning proceeds [9,16,18,30]. The annealing schedule we used is as follows,

$$\alpha = \frac{\alpha_0}{1 + \rho \cdot n}$$

Here α_0 is the initial learning rate n is the number of hidden units added so far and ρ controls how fast the learning rate α decreases.

4.4.6. CCNN Approach—Final results.

We are now in a position to present the final CCNN metamodeling results for the tougher TERSM problem. **Figures 4.19-4.20** show the performance of the CCNN trained with 14,641 training points. In the figures, the solid line is the MSE during training, and the broken line is the MSE for a set of 500 testing points. As expected, simply using more training points improves the performance of the standard CCNN. As **Fig. 4.19** illustrates, using 14,641 training points, (instead of the 500 we used initially) reduces the MSE on the training set from 12,300 to 4,156. As seen in **Fig. 4.20**, using multi-starts with a pool of 5 competing candidates for each unit added, has little effect on the MSE during training, but gives a small improvement in generalization on the testing set. This small improvement in generalization, however, comes at the cost of increased computer time during training. In both cases, the performance rapidly improves as new hidden units are added, leveling off after the algorithm had added about 40 units. Note that in this case the final modeling error for the CCNN is about ±60 emitters. Since the average number of emitters detected over the 14,641 training set is 560, this amounts to an approximate 10% error.

Since TERSM does not take long to execute (≈35 seconds per data point), collecting 14,641 training points was feasible (≈6 days). For more complex simulations, collecting so many training points may not be feasible, and we would like to be able to train the CCNN with far fewer points. With the CCNN training algorithm modified as described in Section 4.4.5, we were able to get acceptable performance using only 3,000 training points. The results in **Fig's 4.21-4.24** clearly show that our final CCNN metamodel is an improvement over the initial CCNN metamodel. Note that **Fig. 4.21** is the same as **Fig. 4.12**, and **Fig. 4.23** is the same as **Fig. 4.14**. These figures give the true TERSM output. **Figure 4.22** is the output of the "final" CCNN metamodel. Comparing this figure to **Fig. 4.21** shows how well the final CCNN metamodel matches the true output. Comparing this figure to **Fig. 4.13** shows how the training algorithm modifications and

the larger training set improve the CCNN metamodeling ability. Similarly, **Fig. 4.24** can be compared to **Fig. 4.23** to see how well the final CCNN metamodel matches the true output of TERSM, and comparing **Fig. 4.24** to **Fig. 4.14** shows how the final CCNN metamodel improves over the initial CCNN metamodel.

4.5. Conclusions.

The most important conclusion we can draw is the following:

• The CCNN works well for simulation metamodeling.

As the response surface becomes more complex, more training points may be required. However, by making certain modifications to the CCNN training algorithm, learning and generalization can be improved, and it should be possible to keep the required number of training points within feasible bounds. In particular, we have shown that,

• Using multi-starts improves generalization;
• Learning rate annealing accelerates convergence.

The real value of metamodeling is the potential speed-up in the ability to get answers. To get one data point using TERSM it takes, on average,

• *34.5 seconds (varies from 10 to 70 seconds)*

To get one data point using a CCNN with 50 hidden units takes, on average,

• CCNN (w/ 50 hidden units): *0.016 second*.

Thus, the CCNN is *2156* times faster and, most importantly, its speed is independent of the complexity of the system it is metamodeling.

In summary, we have developed an efficient neural network approach suitable for metamodeling applications. With the help of more benchmark problems, we hope to further reduce the training data set needed to guarantee satisfactory generalization, and we hope to further improve the CCNN's learning efficiency.
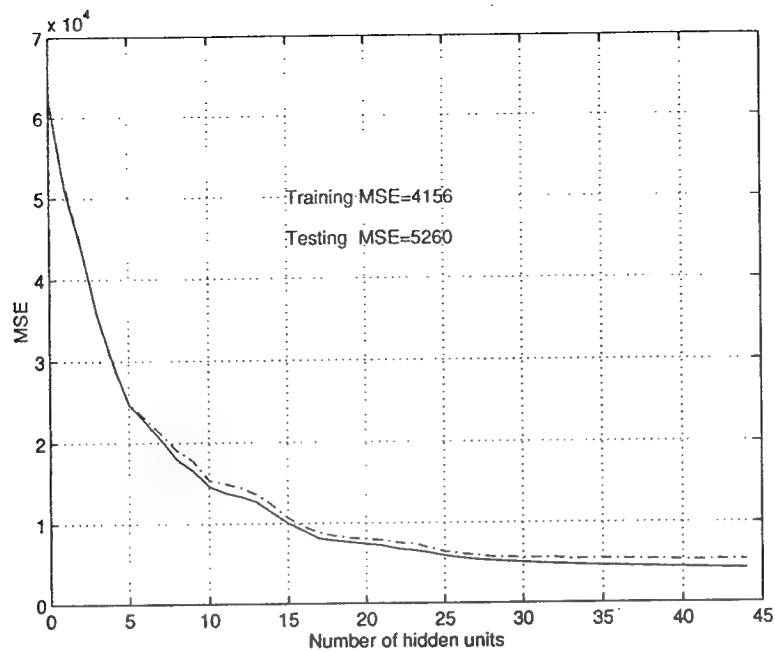
Figure 4.19. Without multi-starts the test MSE is larger than the training MSE.
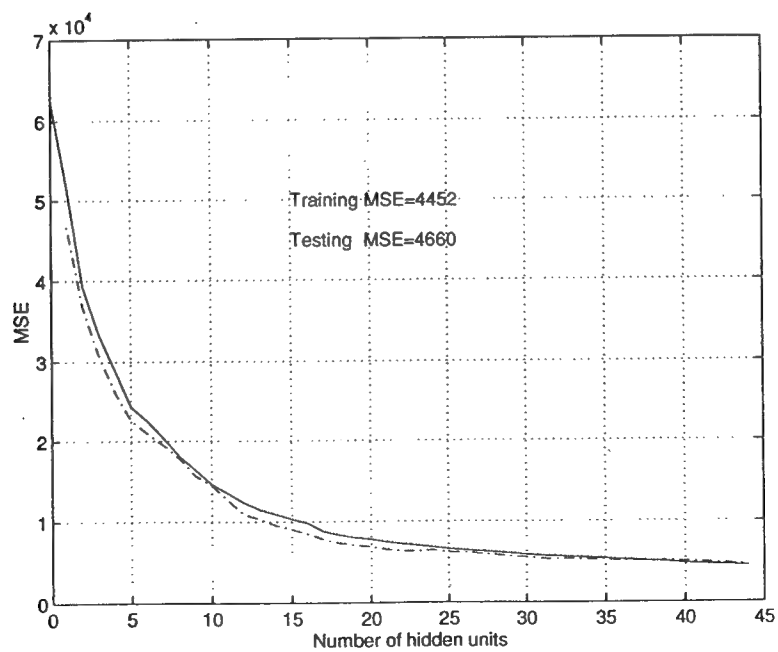
Figure 4.20. With multi-starts the test MSE is reduced.
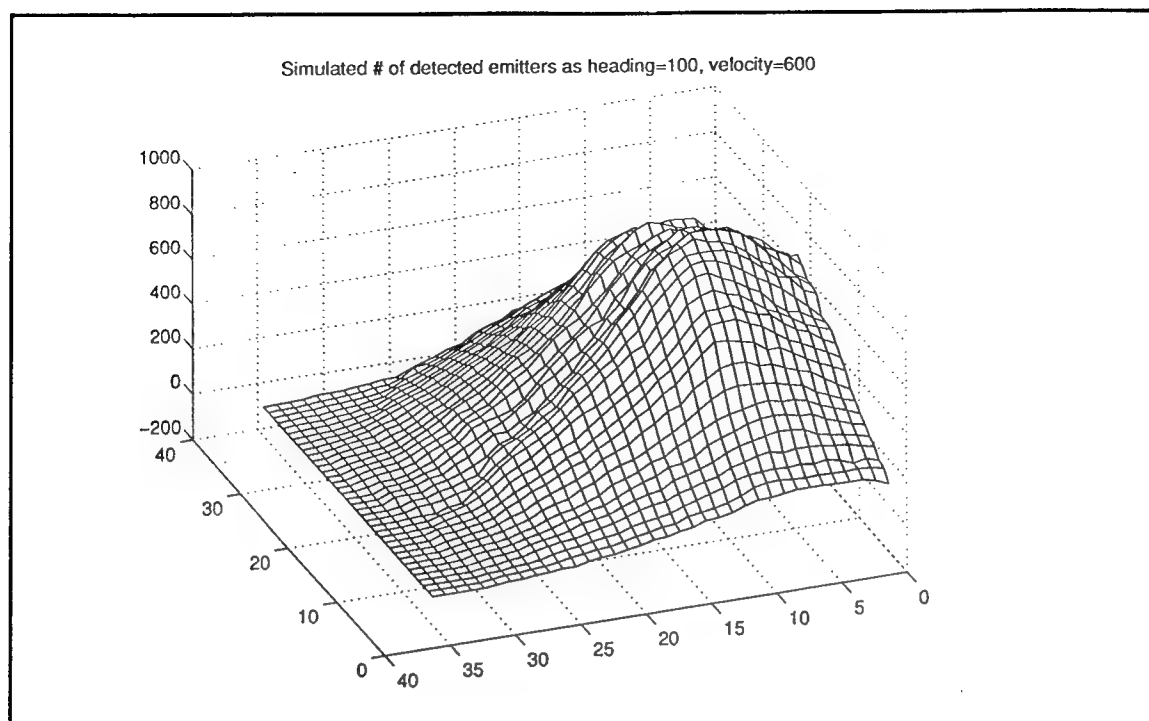
Figure 4.21. $y \sim F(x_0, y_0)$ velocity = 600, $\theta = 100$ (true TERSM output).
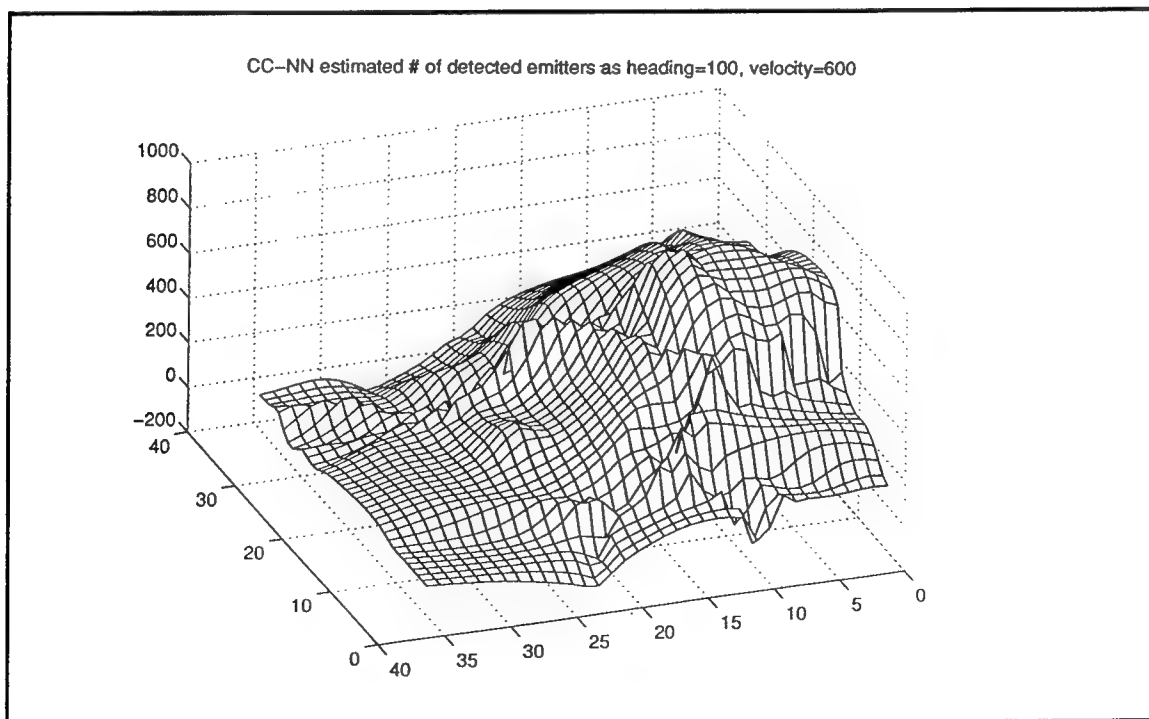
Figure 4.22. $\hat{y} \sim \hat{F}(x_0, y_0)$ velocity = 600, $\theta = 100$ (estimated by modified CCNN trained with 3000 training points).
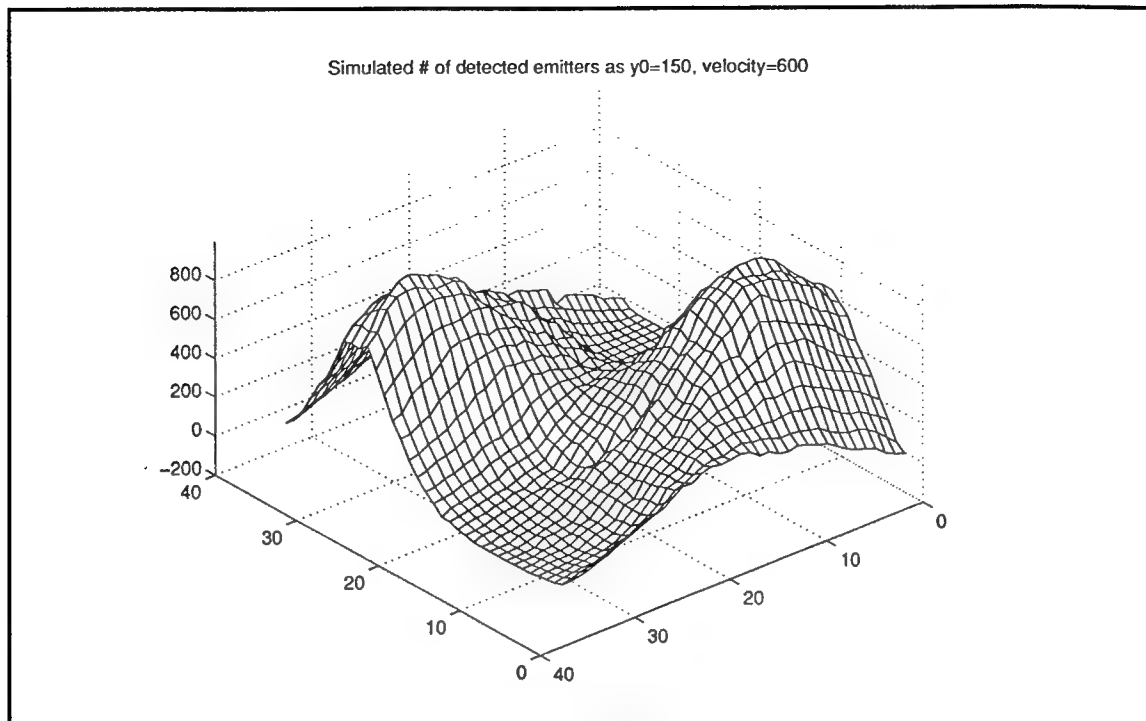
Figure 4.23. $y \sim G(x_0, \theta)$ velocity = 600, $y_0 = 150$ (true TERSM output).
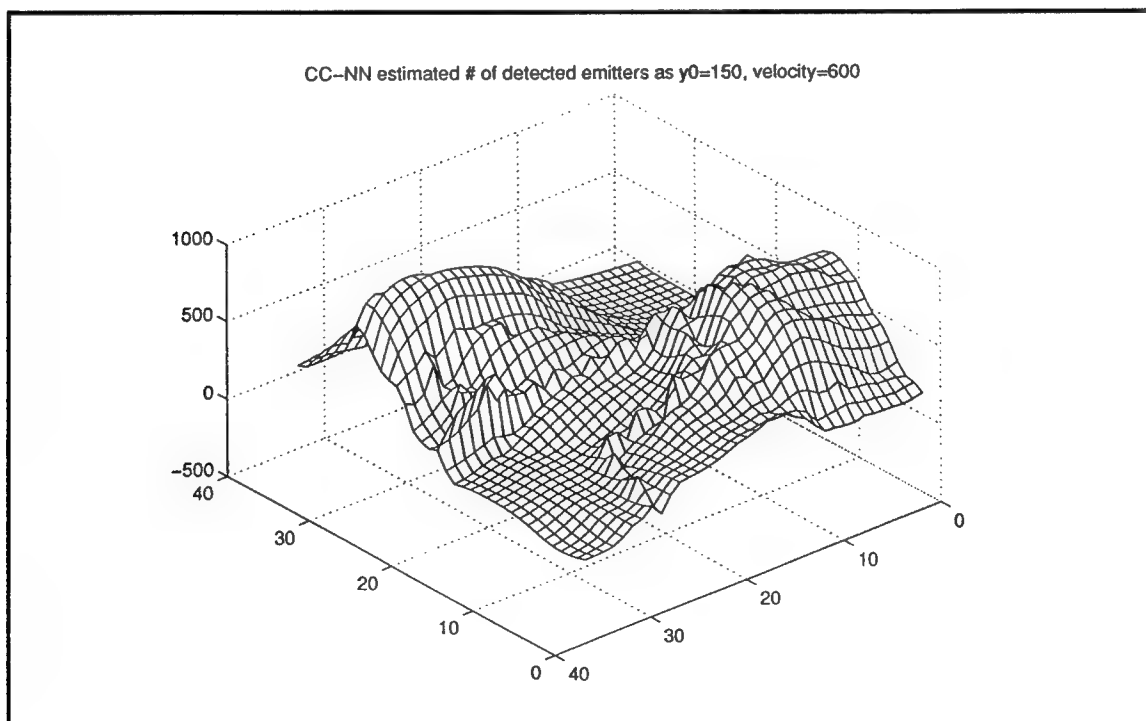
Figure 4.24. $\hat{y} \sim \hat{G}(x_0, \theta)$ velocity = 600, $y_0 = 150$ (estimated by modified CCNN trained with 3000 training points).

5. STOCHASTIC FIDELITY IN HIERARCHICAL COMBAT SIMULATION.

Combat simulation models typically have a hierarchical structure. The top levels summarize the outcome of the overall war, the middle levels summarize the outcomes of individual battles, and the bottom levels simulate the engagements between individual weapon systems. In terms of accuracy and stochastic fidelity, it would be ideal if the highly detailed, high-resolution battalion level engagements could be simulated directly at the top level of the hierarchy. However, since a typical war consists of thousands of battalion level engagements, this would require long execution times. The common practice is to avoid these long execution times by performing the battalion level engagements separately off-line. The results are then aggregated into a weapon-attrition scoreboard, which summarizes the engagements and estimates parameters such as target availability, shooting rate, probability of kill given a hit, target priorities, etc. These parameters are then used to determine the coefficients for a set of nonlinear equations that can be solved numerically to obtain the attrition rate of each weapon type as a function of the average number of such weapons on each side of the engagement. Further reductions in simulation execution time are sometimes achieved by assuming exponentially decaying attrition rates and taking time-averages to summarize the dynamic evolution of the engagement. These approaches were designed at a time when computing power was rather limited (see, for example, Concept Evaluation Model, U.S. Army Concept Analysis Agency (CAA), 1985), and they have been, and continue to be, the workhorse for many military analysts. The problem with these approaches, however, is that unless special precautions are taken, the aggregated data representing the engagements is meaningless. As a consequence, the stochastic fidelity of the overall combat simulation is brought into question.

In this section we discuss the crucial issue of preserving the stochastic fidelity in hierarchical battle simulation models. Using the CAA Concept Evaluation Model as a concrete example, Section 5.1 gives an overview of a hierarchical battle simulation model. A mathematical review of the model is presented in Section 5.2. In Section 5.3 we introduce the Path Bundle Grouping (PBG) approach as a way to maintain stochastic fidelity in hierarchical simulations. The PBG approach is related to the theory of cluster analysis and in Section 5.4 we describe how the adaptive resonance theory (ART) neural network can be employed to automate the task of path bundle grouping. We close in Section 5.5 with a summary.

5.1. Overview of A Hierarchical Battle Simulation Model.

Our work on stochastic fidelity preservation is based on a concrete hierarchical battle simulation model, the COSAGE-ATCAL-CEM model. COSAGE, ATCAL and CEM are the abbreviations of COmbat SAmple GEnerator, ATtrition CALculation and Concept Evaluation Model,

respectively. A version of this model is used in the United States Army Concept Analysis Agency. Conceptually this model can be described in the following **Fig. 5.1.**
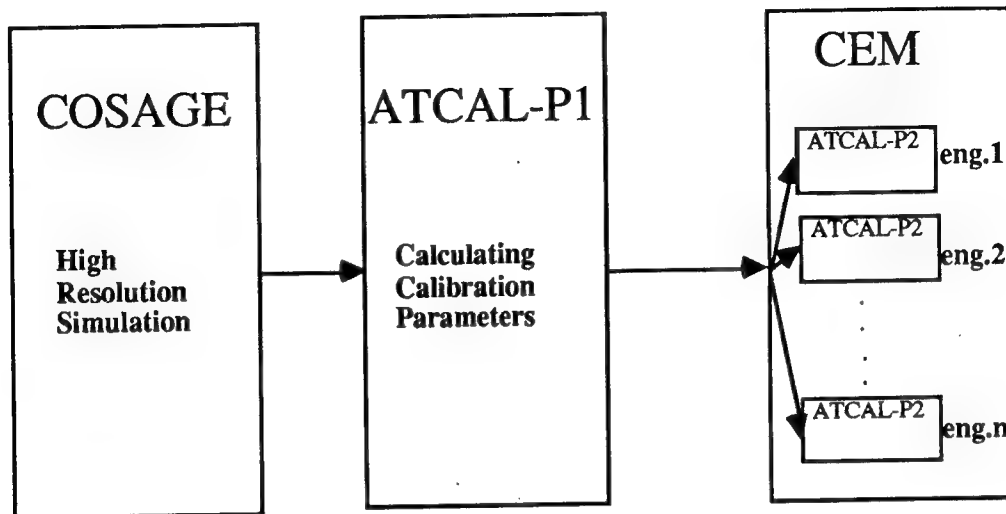
Figure 5.1. A hierarchical battle simulation model.

In the following we explain each component in this system in more detail.

COSAGE:

COSAGE is a high-resolution stochastic simulation model of combat between two forces. Typically, the Blue force is sized as a division, and the Red force is scaled from a fraction of a division to a combined arms army. The model simulates periods (normally 48 hours) of combat and produces expenditures of ammunition by round type and losses of personnel and equipment. Maneuver unit resolution is typically down to Blue platoon versus Red company. In the case of close combat, resolution is to the individual equipment and weapon level. COSAGE is a discrete event simulation model with stochastic phenomena modeled through events and processes. A single COSAGE run uses 53 million U[0,1] random numbers.

For the purpose of easy reference, we list the relevant files in the following:

1. COSAGE Input:

  Forces, and their organizational structures;

  Equipment, weapons, personnel and the characteristics (e.g., strengths, availability, types, densities);

  Munitions types and technical characteristics;

  Probability values used in simulation;

  Priorities, sequences, rules of engagement (tactics and doctrine);

  Sensor definition;

Operation concepts.

2. List of COSAGE input data files:

Air defense sensor; Aircraft munitions; Battery; Category type unit; Counterfire radar; Decision; Equipment; Fire direction center; Forward area rearming and refueling point; Forward observer; High explosives lethal area; Illumination; Mine; Munitions; Orders; Passive detection base; Phased off-line attrition; Posture, environment and mission; Probability of kill; Rules of engagement; Sensor; Smart munitions; Smoke; Sub munitions; System; Tactical aircraft; Target report; Battery type; Battlefield type; Sensor type; Unit; Unmanned air vehicles; Visibility; Weapon.

3. COSAGE Output

> Unit movement;
>
> Ammunition expenditures;
>
> Equipment and personnel losses (Killer/Victim scoreboard);
>
> Shot summary;
>
> Equipment initial density.

4. List of COSAGE output files:

Replication shot summary; Posture shot summary; Killer/victim summary; Direct fire summary; Indirect fire summary; TACAIR/AIRDEF; Target report; Mines; Helicopter/SMARTMUNS report.

Representative combat samples can be produced from COSAGE using skillful and ingenious construction of input files. COSAGE can be used to simulate:

(1) Most types of offensive and defensive operations.

(2) The operations of US, allied, joint, combined (excluding naval), and threat forces from the individual to a combined arms army.

(3) The effects of terrain, weather, and obscurants on combat operations.

(4) The movement of combat and combat support units on the battlefield.

(5) The performance and effects of most current and future weapons and equipment.

(6) The major aspects of target acquisition such as ground surveillance radar, forward observers, flash and sound sensors, counter mortar and counter battery radar, and remotely piloted vehicles.

(7) The direct fire battle effects due to small arms, tanks, fighting vehicles, antitank guided missiles, and mines.

(8) The indirect fire effects from tube artillery, mortars, and multiple launch rockets --- to include the effects from conventional and improved conventional munitions, precision guided munitions, laser guided munitions and mines.

The objectives of COSAGE are:

    (1) To calculate expected ammunition expenditures and equipment/personnel losses for both sides.

    (2) To provide a record of killer/victim relationships to the ATCAL for calibration.

    (3) To provide a record of losses and expenditures to the simulation post processors.

Generally, six different combat operations are modeled for a study using one or more of the following types of terrain.

- Terrain Types:

    A. Flat to gently rolling with minimum obstacles.

    B. Gently rolling terrain with some obstacles and vegetation.

    C. Mountainous with steep slopes and/or dense forestation or swamps.

- Combat Operation Types:

    (1) Blue Intense Defense (I): The blue division in a prepared defense against an attacking Red force.

    (2) Blue Delay (D): The Blue division conducting a delay or a defense on alternate or successive defense positions against an attacking Red force.

    (3) Blue Hasty Defense (H): The Blue division in a hasty defense against an attacking Red force.

    (4) Static (L): The Blue division and a Red force are at parity and are both in defensive positions. Both sides are conducting patrols, probes, and reconnaissance.

    (5) Blue Attack (F): Multiple Blue divisions conducting an attack against a Red division in a prepared defense.

    (6) Red Hasty Defense (N): Multiple Blue divisions conducting an attack against a Red division in a hasty defense.

The results of the division combat simulations are called combat samples. Combat samples represent the expected results, during a theater campaign, for division combat for 2 days of the posture simulated. The combat sample is not intended to represent the first high-intensity period nor the last period when intensity is expected to be low.

Combat samples are analyzed to ensure that the military aspects of combat are adequately and faithfully portrayed in the model. The ammunition expenditures and equipment losses for both the Red and Blue forces are determined, and killer/victim relationships are established.

ATCAL:

A key function of any combat simulation is the calculation of losses of equipment and personnel in the engaged forces. In simulations of small-scale engagements, this can be accomplished through a detailed treatment of all shooters and their potential targets, with each firing opportunity examined as to its feasibility and outcome. The high-resolution simulation (resolved down to the interactions between individual weapons) can be done readily at battalion level but only with great computer time and resource penalties at division level. Although this approach gives the most realism, it cannot be considered for the large-scale combat occurring at theater level. Here one must use attrition equations to compute the interactions between Blue and Red weapons. These attrition equations are of various forms, all attempting to relate numbers of shooters and numbers of targets to losses in engagements. Attrition equations always contain parameters which are difficult to determine, such as allocations of fire to various target types or maximum rate at which a given type of shooter can kill a given type of target. They also make use of aggregation, the grouping together of weapons and targets in some logical way in order that the losses inflicted by each of the groups upon each group in the opposing force can be computed. The Concepts Evaluation Model (CEM) aggregates the weapons into Blue and Red heavy armor (tanks), light armor (APCs), soft (dismounted personnel), artillery, helicopter, and fixed-wing categories. Other theater models aggregate into vehicle types: several tank types, several APC types, several artillery types, etc. When the parameters of an attrition equation are chosen in such a way that aggregated combat results agree with the results of a high-resolution simulation, for a particular mix of weapons on the two sides, the equation is said to be calibrated. A large part of chronic dissatisfaction with large-scale simulations is due to the failure of attrition equation results (with a fixed set of parameters) to track well with high-resolution results (or with judgmental expectations) as the composition of the forces is varied.

ATCAL is a method for calibrating a set of attrition equations to the results of sample high-resolution simulations. It uses auxiliary equations to feed the main attrition equations, modifying their parameters and thereby accounting for considerably more battlefield detail than heretofore. This added flexibility permits better portrayal of the results of force variations. The method uses high resolution results and provides useful side information in addition to the loss-by-cause table (commonly refereed to as a killer-victim scoreboard). This side information is the allocation of fire among all shooter and target types, the expenditures of ammunition, the relative importance of all weapons, and the force ratio in the engagement.

The starting point of the new methodology is a pair of new attrition equations, one for point fire and one for area fire. Both equations compute losses by cause and are therefore fundamentally different from attrition equations which operate with aggregated firepower. The point-fire equation has two main calibration parameters: kills per round fired at the target and availability of a target to

a shooter. These two parameters, each depending upon shooter and target type, can be uniquely determined from two main output matrices of the high-resolution simulation: the firing matrix (shots fired by each shooter type against each target type) and the attrition matrix (or killer-victim scoreboard). Calibration for the point fire part consists of using the ATCAL methodology "backward" to determine the two parameters. To determine losses for a force of some different composition from the calibration force the ATCAL methodology is run "forward". The area fire equation utilizes two main calibration parameters also: the kills per round (given that the round is associated with the target) and a response factor for each tube type, which states the volume of fire delivered to satisfy the demand for fire (the demand being an internal ATCAL calculation). There are no time steps in ATCAL—the entire engagement is treated as a single time step and average numbers of participants are used, instead of starting numbers, to produce a decrease in fire as losses go up.

Discussion of ATCAL: The ATCAL model is an implementation of a method which employs hundreds of interconnected equations. We are to describe the main equations of ATCAL and the reasoning behind; and to show how the equations are coupled together. The coupling is different in each of the two distinct parts of ATCAL. The two parts, called Phase I and Phase II, are what was referred to above as running the model backward and forward—for calibration of parameters and for prediction of results when new forces are employed. In our report, we concentrate on ATCAL phase II.

Importance of Weapons: Fire allocation is part of the ATCAL process. Since allocation goes preferentially to the more important targets, a means of quantifying the importance of targets is a necessary first step in fire allocation. The importance of weapons comes from a nonlinear operation on the killer-victim scoreboard and uses only the scoreboard itself. Since the scoreboard is a key element in ATCAL, the importance can be readily computed within ATCAL for the engagement at hand.

Vehicles and their average number: Each vehicle, used to denote a killable entity on the battlefield, can be both shooter and target and, in general, each will have several weapon types on it. The average numbers of vehicles of each type in the engagement are used in the attrition equations to produce a dynamic model which responds appropriately to changes in engagement length. For an engagement with heavy attrition of some systems, it is very wrong to use the number present at the beginning in computing fire allocations. Fire allocation to a target type should decrease as the target is depleted. Similarly, the fire from that depleted vehicle type should decrease. Using an average number, which can assume arbitrarily small values, in the attrition equations is a way of taking a vehicle out of the picture in response to its depletion. The straight

line average cannot assume arbitrarily small values, therefore the average is computed from an assumed exponential decrease in weapon count during the course of the engagement.

Suppose the starting number of weapons is N (see **Fig. 5.2**). After time t, the remaining number becomes $N(t) = Ne^{-\lambda t}$. Therefore, if we denote the attrition by DN, the average number of weapons during the engagement period T, $\bar{N} = \frac{1}{T} N \int_0^{} e^{-\lambda t} dt$ can be expressed as

$$\bar{N} = -\Delta N / \ln(1 - \frac{\Delta N}{N}) \tag{5.1}$$
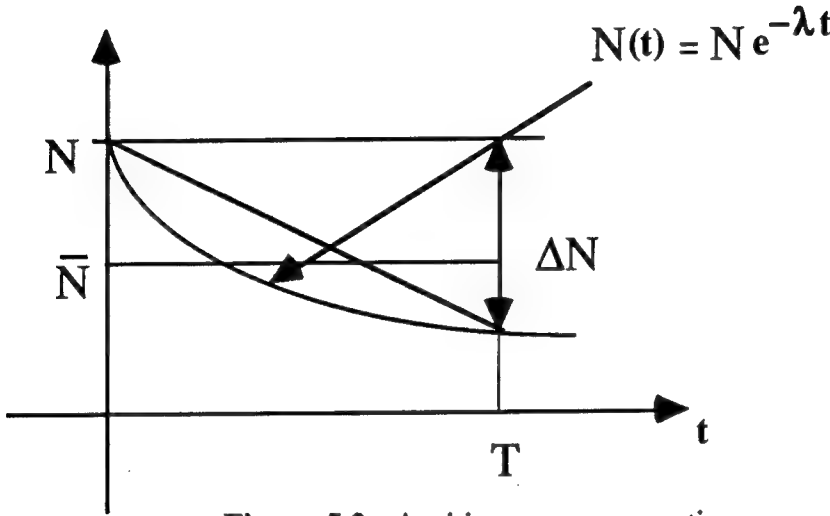
Figure 5.2. Attrition process over time.

Note that in the above equation ΔN and $\bar{N}$ depend upon each other in an inverse sense (an increase in one results in a decrease of the other), it is clear that there must be a solution in which the two are in equilibrium. Therefore, in ATCAL phase II, we take $\bar{N} = N$ initially and iterate to solve for DN.

We note that there are actually some justifications for the exponential decay assumption for the point fire. Denote by R and B the average strength of the red army and blue army, respectively. Then according to the Lanchester equation,

$$\begin{cases} \dot{R} = -c_r * B \\ \dot{B} = -c_b * R \end{cases}$$

from which we can get

$$\begin{cases} R(t) = R_0 * e^{-\sqrt{c_r * c_b} t} \\ B(t) = B_0 * e^{-\sqrt{c_r * c_b} t} \end{cases}$$

with

$$\frac{B_0}{R_0} = \sqrt{\frac{c_b}{c_r}}.$$

Target priorities: Each weapon or round type must have its targets prioritized so that the model can compute allocations of fire to targets for point fire or allocations of rounds by type for area fire. Target priority is computed as the product of kills per round and target importance. Thus a target only becomes lucrative to a weapon if it is both important in itself and more easily killed than other targets available to the weapon.

Attrition equation for direct fire: Consider a property of a shooter-target pair to be target availability, defined as the fraction of time a particular target (of a type) can be fired upon by a particular shooter (of a type). The quantity can be thought of as averaged over all shooters and targets (of the types in question). Denote this quantity by A_{ijk} --- the fraction of time that k-th target is available for j-th weapon of i-th vehicle. If there are N_k targets of type k, the fraction no target is available is $(1-A_{ijk})^{N_k}$, so $1-(1-A_{ijk})^{N_k}$ is the fraction of time at least one target is available. Now each weapon j of all the N_i shooters is capable of firing $(RATE)_{ij}$ rounds during the engagement, and it will fire at vehicle k if no other target with higher priority is available. If the vehicles with higher priorities are index by k', and the probability of kill per round is P_{ijk}, then we can write the attrition equation for point fire as

$$(\Delta N_k)_{ij} = \bar{N}_i (RATE)_{ij} P_{ijk} [1-(1-A_{ijk})^{\bar{N}_k}] \prod_k [1-A_{ijk'}]^{\bar{N}_{k'}} \tag{5.2}$$

Rewrite equation (5.2) as

$$\Delta N_k = (1-e^{-\Delta N_k/\bar{N}_k}) N_k \tag{5.3}$$

Given the starting number of vehicles, the calibrated parameters $(RATE)_{ij}, P_{ijk}, A_{ijk}$, we can then solve the nonlinear equations (5.2) and (5.3) by iteration to obtain ΔN_k. The importance index k' changes along with each iteration. This process is summarized as the following steps:

Iterative Algorithm for DNk and Ñk (ATCAL II):

Step 0: Set $\overline{N_k^0} = N_k, (t=0)$;

Step 1: Calculate $(\Delta N_k^{t+1})' = f(\overline{N_i^t}, \overline{N_k^t}, \overline{N_{k'}^t})$ using (5.2);

Step 2: $\Delta N_k^{t+1} = [1-e^{-(\Delta N_k^{t+1})'/\overline{N_k^t}}] N_k$;

Step 3: Calculate $\overline{N_k^{t+1}} = \overline{N_k^t} * \Delta N_k^{t+1} / \Delta N_k^t$;

Step 4: Check if $|\overline{N_k^{t+1}} - \overline{N_k^t}|/N_k < \varepsilon$ or $t \leq 15$. If not, set $t = t+1$ and go to step 1.

Remarks:

1. Note that the purpose of the iterations above is to solve the nonlinear equations, not to simulate the evolution of the system over time. In another word, there is no time steps in ATCAL; the entire engagement is treated as a single time step and average numbers of participants are used, instead of starting numbers, to produce a decrease in fire as losses go up.

2. The weapon importance changes during the course of the iteration. This is due to the fact that the importance of a weapon is unknown to us and it depends on the number of kills and the importance of kills inflicted by this weapon. Therefore, as the attrition happens, the importance of the weapons is also adjusted. Eventually, the weapon importance converges because the attrition values converge to their solution.

Area Fire Attrition Equations: As the model sequentially processes the weapon type on each shooting vehicle, it encounters an indicator which tells it whether the weapon is to be processed with point-fire or area-fire. The attrition equations are different for these two types of fires and so are all the parameters that go into them.

The problem for area-fire is to account for (a) the amount of firing that is to be done; (b) the apportionment of the firing among the different round types; (c) the effects of the firing on the target arrays.

The computation of the attrition due to area-fire involves many intermediate steps, which we do not describe here. We merely list the attrition equation in the following and note that the principle of solving the nonlinear equations for area-fire is no different from that for point fire. The area-fire attrition equation is

$$(\Delta N_k)_{ij} = (RSPNS)_i \ (DEMAND)_i \ (Allocation)_{ij} \ (FRAC)_{ijk} \ (\bar{N}_k / N_k) L_{ijk} \qquad (5.4)$$

where $(RSPNS)_i, L_{ijk}$ are calibration parameters obtained from ATCAL phase I, while $(DEMAND)_i$, $(Allocation)_{ij}$, $(FRAC)_{ijk}$ are obtained from the above and another calibration parameter $(BIAS)_{ij}$ in the intermediate steps. With the assumption that the number of vehicles decreases exponentially, we can still use equation (5.2) and thus solve the nonlinear equations (5.3) and (5.4) recursively to obtain the attrition due to area-fire. The flow chart of the ATCAL Phase 2 main cycle is shown in **Fig. 5.3**.

5.2. Mathematical Review of CEM and STOCEM.

First a comment on the nonmonotonic behavior of CEM. The nonmonotonic behavior or "structural variance" has occurred in the CEM, stemming from the use of decision thresholds; a slight increase in strength of one side may alter the force ratio that is compared to a decision threshold, resulting in a different allocation of forces and a significantly different outcome. This phenomenon deserves much more research but it is beyond the scope of this project. Next we try to review the underlying philosophy of the COSAGE-ATCAL-CEM from its mathematical formulation. We start with the ultimate goal of a hierarchical battle simulation model.

What do we really want from CEM? What we wish to obtain through a battle simulation model is the high resolution simulation estimates for the mean (and higher moments of, if possible) attrition at the theater level. We denote the mean attrition for the "throughout high resolution simulation (THRS)" as

$$E[g(w)]$$

where $g(w)$ is the sample THRS output with seed w.

The COSAGE-ATCAL-CEM approach to this problem can be divided into two steps:
a) Run COSAGE to get samples of "calibration" parameters such as
   probability of kills per round $P(w)$;
   availability (w);
   firing rate $R(w)$;
   response factor $F(w)$;
   bias $B(w)$;.
b) Use a Lanchester-like attrition equation to calculate the mean attrition, i.e., approximate $E[g(w)]$ by

$$DN=f(P, A, R, F, B) \qquad \text{(ATCAL II)}$$

We now discuss the limitations of the CEM approach.

Phase 2 main cycle
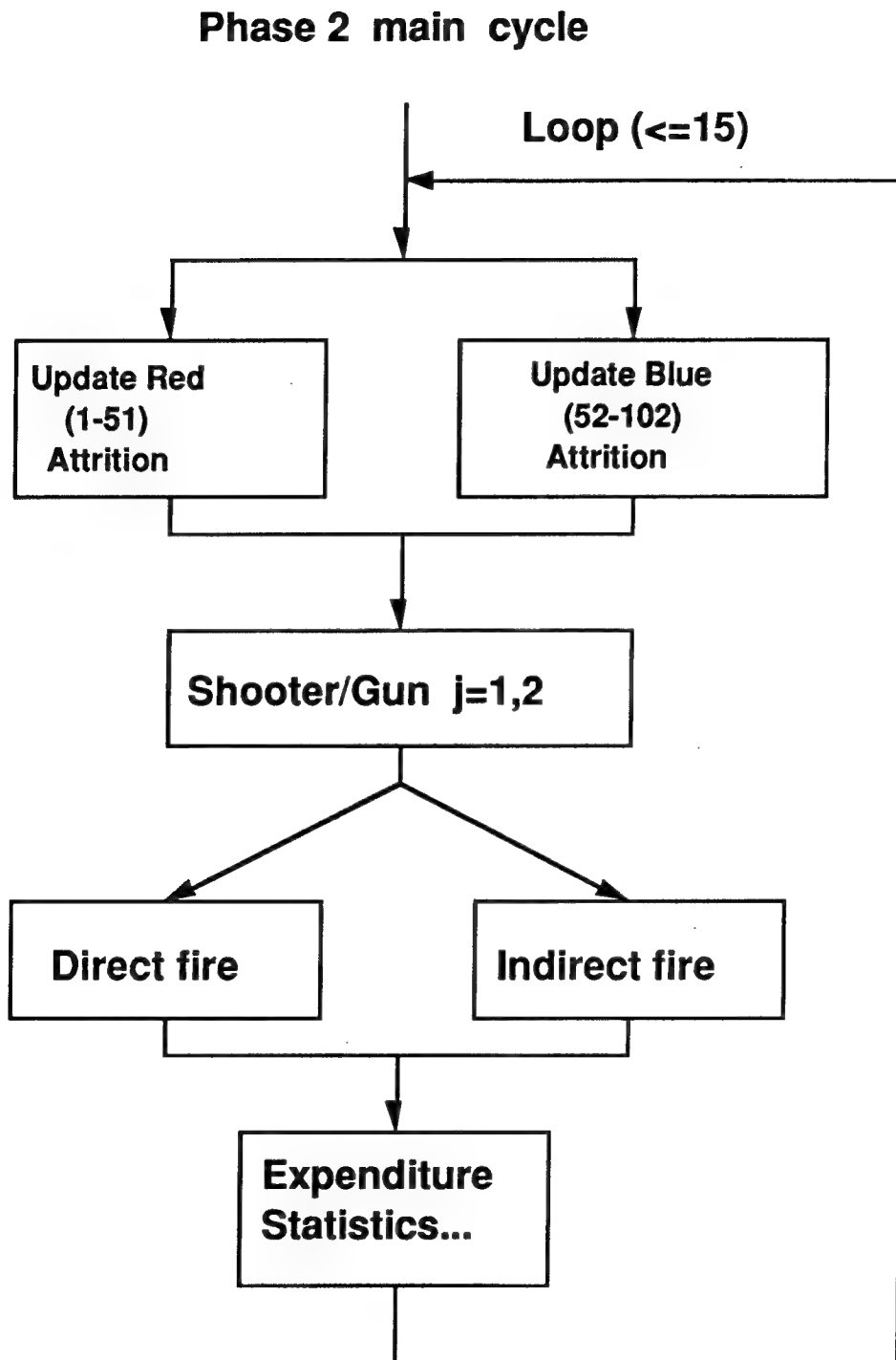
Figure 5.3. Flow chart for ATCAL PHASE II.

Current CEM model has fixed engagement duration and can not be interrupted. More seriously, although CEM accepts different initial mixtures as input, the execution of ATCAL Phase II inside CEM uses the output from COSAGE with a single initial posture. This is due to the fact that setting up an initial mixture for COSAGE takes a long time and a lot of effort. Needless to say this situation is not satisfactory and more research is needed to handle this. Finally, as we explained before, in ATCAL Phase II one averaged the attrition over the whole duration of the engagement and assumed exponential decrease of the attrition. Since each engagement always consists of an initial search period with not much firing, an intensive firing period and an ending period when one side or both sides decide to quit the engagement, to average over time is hardly descriptive of the attrition process. In summary, the current CEM approach has the following limitations:

    (1)   fixed engagement duration,

    (2)   no interruptibility,

    (3)   fixed initial mixture for a given posture in COSAGE..

    (4)   averaged over time

Next we discuss the central issue of our report, the preservation of the stochastic fidelity in CEM model. The following are the two major points in these issues.

- COSAGE runs generate drastically different scenarios, and using the overall mean of the calibration parameters over all the scenarios as the input parameter to ATCAL Phase 2 may not give good attrition estimates.

- COSAGE sample shows that the attrition dynamics are clearly different during different engagement periods. So use of the average over time may not be appropriate.

These points have been observed and there have been attempts to overcome some of these problems. One such approach being considered is the STOchastic CEM (STOCEM) approach. We now discuss this approach.

The stochastic fidelity issue has been noticed in the battle simulation society and a certain amount of effort has been devoted to resolve the issue. Stochastic CEM (STOCEM) is one such approach. The basic idea of STOCEM is to use each COSAGE sample as the input to the ATCAL Phase 2. The output of the CEM is considered as one sample of the stochastic output. The final result is the distribution of, say, the attrition, rather than their mean values as in the case of CEM. 10-12 replications from COSAGE is considered good enough in the current version of STOCEM. For each replication, STOCEM still uses the original calibration procedure. Experimental results show that sometimes the original CEM outputs lie out of the confidence interval of the corresponding STOCEM results.

We think STOCEM is a commendable effort. However, since for each replication STOCEM still uses the original calibration procedure, one has to ask the fundamental question whether the

mean value-based Lanchester Law is valid for samples. To answer this question we need to review the mathematical description of the STOCEM approach.

STOCEM begins by running COSAGE to get samples for "calibration" parameters such as probability of kills per round $P(w)$; availability (w); firing rate $R(w)$; response factor $F(w)$; bias $B(w)$; others. Then, STOCEM uses a Lanchester-like attrition equation to obtain samples of the attrition, i.e., approximate $g(w)$ by $f(P(w), A(w), R(w), F(w), B(w))$. Finally, when a mean value is desired, STOCEM estimates $E[g(w)]$ by averaging over $f(P(w), A(w), R(w), F(w), B(w))$.

The major question we have for the STOCEM approach is whether the Lanchester attrition equations are valid for samples of COSAGE. One could argue that there is considerable averaging in COSAGE over many possible battle scenarios already, so the Lanchester Law could be applied to the COSAGE output. However to average over scenarios generated from a given initial mixture along time is different from averaging over scenarios with more rational weights. This is because the weights attached to the former approach are determined by the battle evolution along time. Such an average could be more wrong than uniform weighted average, let alone a rational choice of the weights. The latter should be what we would like to achieve.

In summary we have the following observations regarding the STOCEM approach.

• Lanchester-like equations are based on the idea that average attrition is proportional to enemy's average strength, which makes sense only for the mean values.

• Note $E[f(X)] \neq f(E[X])$ where f is ATCAL II which does not make much sense to the samples X!

• Even if we know the distribution of P, A, etc., we still don't know how to incorporate them (or just the second moment) into the attrition equation to reflect their impact on $E[g(w)]$

• Over the range of starting points that ATCAL II must address, the calibration parameters remain constant for the posture.

• CEM 12 hours engagement is actually a kind of average over (unspecified) phases of a 48 hour combat process.

Mathematically the fundamental issue with the preservation of the stochastic fidelity lies in the inequality $E[f(x)] \neq f(E[x])$ with nonlinear $f(\cdot)$. We give several examples to illustrate this inequality. Our first example is extremely simple:

Example 1: $\qquad f(x) = x^2 \qquad \Rightarrow \qquad E[x^2] = s^2 + E[x]^2$

Our second example is closer to the equations used in ATCAL for computing the attrition.

Example 2: $f(x) = a^x, \quad 0 < a < 1$

Note that in ATCAL we need to use the power function to compute probabilities involved in the attrition calculation. In the following we illustrate the inequality $E[f(x)] \neq f(E[x])$ for several different distributions for x.

Example 2a.

$$\Pr(X = 1) = p, \quad \Pr(X = 0) = 1 - p$$
$$E[X] = p$$
$$f(E[X]) = a^p$$
$$E[f(X)] = pa^1 + (1 - p)a^0 = pa + 1 - p$$

Example 2b.

$$\Pr(X = k) = q^k p, \quad q = 1 - p, \quad k = 0,1,\ldots$$
$$E[X] = \frac{q}{p}$$
$$f(E[X]) = a^{\frac{q}{p}}$$
$$E[f(X)] = pa^0 + qpa^1 + q^2 pa^2 + \ldots = \frac{p}{1 - qa}$$

Example 2c.

$$\Pr[X = k] = \frac{l^k e^{-l}}{k!}, \quad for\ k = 0,1,\ldots$$
$$E[X] = l$$
$$f(E[X]) = a^l$$
$$E[f(X)] = \frac{l^0 e^{-l}}{0!}a^0 + \frac{l^1 e^{-l}}{1!}a^1 + \frac{l^2 e^{-l}}{2!}a^2 + \ldots = e^{-l}e^{la} = e^{-l(1 - a)}$$

5.3. Path Bundle Grouping Approach.

Since different random seeds in COSAGE shall result in different paths, the correct thing to do is to group the results of COSAGE into path bundles, $S_1, S_2, \ldots, S_n$, and calculate

$$E[g] = E[g|S_1]*Pr(S_1) + E[g|S_2]*Pr(S_2) + \ldots + E[g|Sn]*Pr(Sn)$$

For each path S_i, we can get the calibration parameters P_i, A_i, etc. and then use $f(P_i, A_i, \ldots)$ to approximate $E[g|S_i]$.

Professor S.M. Robinson (University of Wisconsin) made the following remarks in [25]: "Note that this use of an expected value performance measure is not at all the same thing as the common use of "expected value models" in which stochastic elements are individually and

systematically replaced by their expected values. That procedure is invalid as a method for modeling anything, since the outcome can not be reliably related to the average of the outcomes under the individual scenarios, or to any other quantity of interest. Rather, the expected value performance measure that we are using corresponds to use of a Monte Carlo simulation process, but (as we shall see below) with a certain degree of increased structure."

In the following we use a queuing model example to show that different random seeds result in different function $g(w)$.

The example performance function we use is the mean waiting time of the 5th customer in an M/M/1 queue. For each random seed, we get the samples of the interarrival time $a_1(w), a_2(w),...,a_5(w)$ and the samples of the service time $s_1(w), s_2(w),...,s_5(w)$. The waiting time of the 5th customer is the sum of the service time of all the previously arrived customers who are in the same busy period as itself. So if we define Si as the scenario that i customers are in front of the 5th one and are in the same busy period, then we have

$$W_5(\vec{a}(w), \vec{s}(w)|S_i) = f_w(\vec{a}(w), \vec{s}(w)|S_i) = \sum_{k=5-i}^{4} s_k$$

Hence different random seeds not only result in different $\vec{a}(\omega), \vec{s}(\omega)$ but also different functions. We show pictorially in **Fig. 5.4** that different random numbers could generate different scenarios.
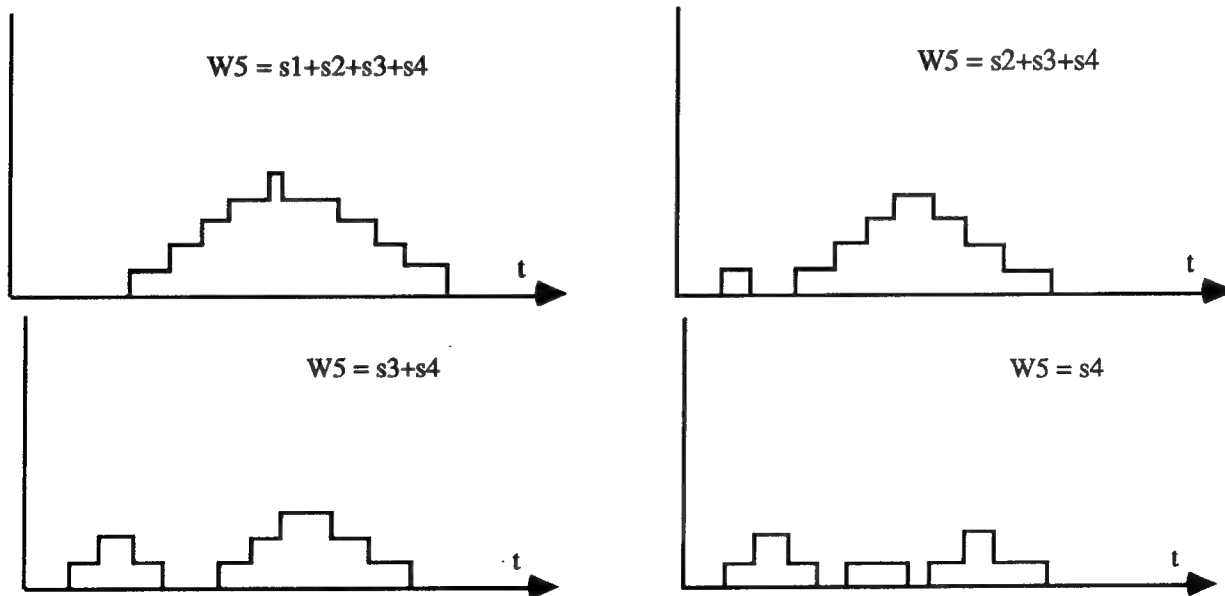
Figure 5.4. Four scenarios for a simple queuing sample path.

Now we use a simple queuing example to illustrate the scenario grouping idea. Consider a M/M/1/K queue with utilization r. The probability of buffer full is calculated by the mean formula

$$P_K(\rho) = \frac{(1-\rho)\rho^K}{1-\rho^{K+1}} \tag{5.5}$$

This formula is valid only when r is a constant. When used in a simulated model, r has to be estimated accurately. Using (5.5) with a sample of r is not justified.

We compare this queuing example with the battle simulation model CEM:

M/M/1/K Queue	CEM
$P_K(\rho) = \dfrac{(1-\rho)\rho^K}{1-\rho^{K+1}}$	
	$DN = f(P, A, ...)$
r	$P, A, ...$

The question is : Can we use (25) when r is a random variable? Suppose we have a MMPP/M/1/K queue shown in **Fig. 5.5**. We want to estimate r and use it as the calibration parameter and then use equation (5.5) to estimate P_K. Suppose also that we have chosen $K = 8$, $m = 10$, and obtained eight estimates of r: 1,2,3,4,5,6,7,8. Now we estimate P_K using four different schemes:

1) Treat all the estimates as one group (a la CEM), i.e. calculate the mean of r, 4.5, then (5.5) gives $P_K = 0.93 * 10^{-3}$

2) Divide the estimates into two groups: {1,2,3,4} and {5,6,7,8}, compute their respective mean value: 2.5, 6.5 and then use (25) to get

$$P_K = P_K(2.5) * 0.5 + P_K(6.5) * 0.5 = 5.7 * 10^{-3}$$

3) Divide the estimates into four "scenario groups": {1,2}, {3,4}, {5,6}, {7,8} and then use (5.5) to get

$$P_K = P_K(1.5) * 0.25 + P_K(3.5) * 0.25 + P_K(5.5) * 0.25 + P_K(7.5) * 0.25 = 7.7 * 10^{-3}$$

4) Divide the estimate into eight groups (a la STOCEM), i.e., plug each estimate into (5.5) and then compute the average. In this case we get $P_K = 8.2 * 10^{-3}$

The question is: Which scheme is "more correct"?

$$\rho = \lambda/\mu$$
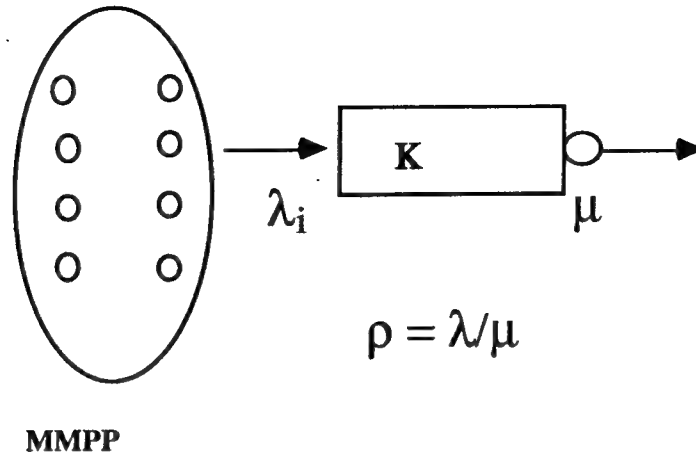
MMPP

Figure 5.5. MMPP/M/1/K Queue.

Suppose the arrival process is modulated by a two-state continuous time Markov chain. The mean sojourn time of each modulating state is 50, the arrival rate associated with each modulating state is 2.5, 6.5, respectively. By simulation, $P_K = 5.5*10^{-3}$. So method 2) above gives a good estimate (Because 3 and 4) did not adequately "smooth" the samples and 1) fails to convey the impact of the two scenarios).

Suppose the arrival process is modulated by a four-state Markov chain. The sojourn time of each modulating state is still 50, the arrival rate associated with each modulating state is 1.5, 3.5, 5.5, 7.5, respectively. By simulation, $P_K = 7.6*10^{-3}$. So method 3) above gives a good estimate.

Let us now return to the COSAGE-ATCAL-CEM model. The key question is how to do the grouping. The answer of this question depends on the computational power available to us. Basically we could have four levels of grouping.

- Level 1: Group according to time: first, second, and last period of a battle;
- Level 2: Group according to the ATCAL I output;
- Level 3: Group according to several key quantities in COSAGE output;
- Level 4: Clustering analysis of COSAGE output (using large neural net).

Note that in this approach we also need to estimate the probability of each group. For example, in dealing with different initial mixture, we need to assume these probabilities. To estimate these probabilities from simulated data we need to perform clustering analysis. We consider the following clustering analysis algorithm via ART2 neural network.

5.4. Path Bundle Grouping of COSAGE Output using ART2.

If each COSAGE Output path can be represented by a vector, the Path Bundle Grouping problem is just a vector clustering problem, which can be tackled by Adaptive Resonance Theory (ART)[17,19,11]. We first try to represent the COSAGE Output paths by the binary vectors through preprocessing the source simulation data. Then we use the so-called ART2 neural network to do the path clustering. The results show that this approach works quite well with arbitrary vector input order.

5.4.1. Overview of ART2.

Adaptive Resonance Theory (ART) was developed by Carpenter and Grossberg [1]. The ART2 is designed to cluster input vectors with less sensitivity to the order of presentation. The architecture of ART2 is shown in **Fig. 5.6**. In Figure 5.6, $f(x)$ is the activation function.

$$f(x) = \begin{cases} x & \text{if } x \geq \theta \\ 0 & \text{if } x < \theta \end{cases}$$

This function suppresses any components of the vectors of activations that fall below the user selected value θ.

The F1 layer consist of six types of units (the W, X, U, V, P, and Q units). There are n units of each of these types (where n is the dimension of an input pattern). Only one unit of each type is shown in the figure. The symbols on the connection paths between the various units in the F1 layer in figure 1 indicate the transformation that occurs to the signal as it passes form one type of unit to the next. The symbol --> indicates normalization; i.e., the vector q of activations of the Q units is just the vector p of activations of the P units, normalized to approximately unit length.

The units at F2 layer compete in a winner-take-all mode for the right to learn the input pattern. The activation of the winning F2 unit is d, where 0<d<1. The learning occurs only if the top-down weight vector for the winning unit is sufficiently similar to the input vector. The tests for reset will be discussed later.

Algorithm of ART2

1. The input signal s continues to be sent while all of the actions to be described are performed. At the beginning of a learning trial, all activations are set to zero.
2. First cycle of updating the F1 layer.

$$u_i = 0 \qquad x_i = \frac{s_i}{\| s \|}$$

$$w_i = s_i \qquad q_i = 0$$

$$p_i = 0 \qquad v_i = f(x_i)$$
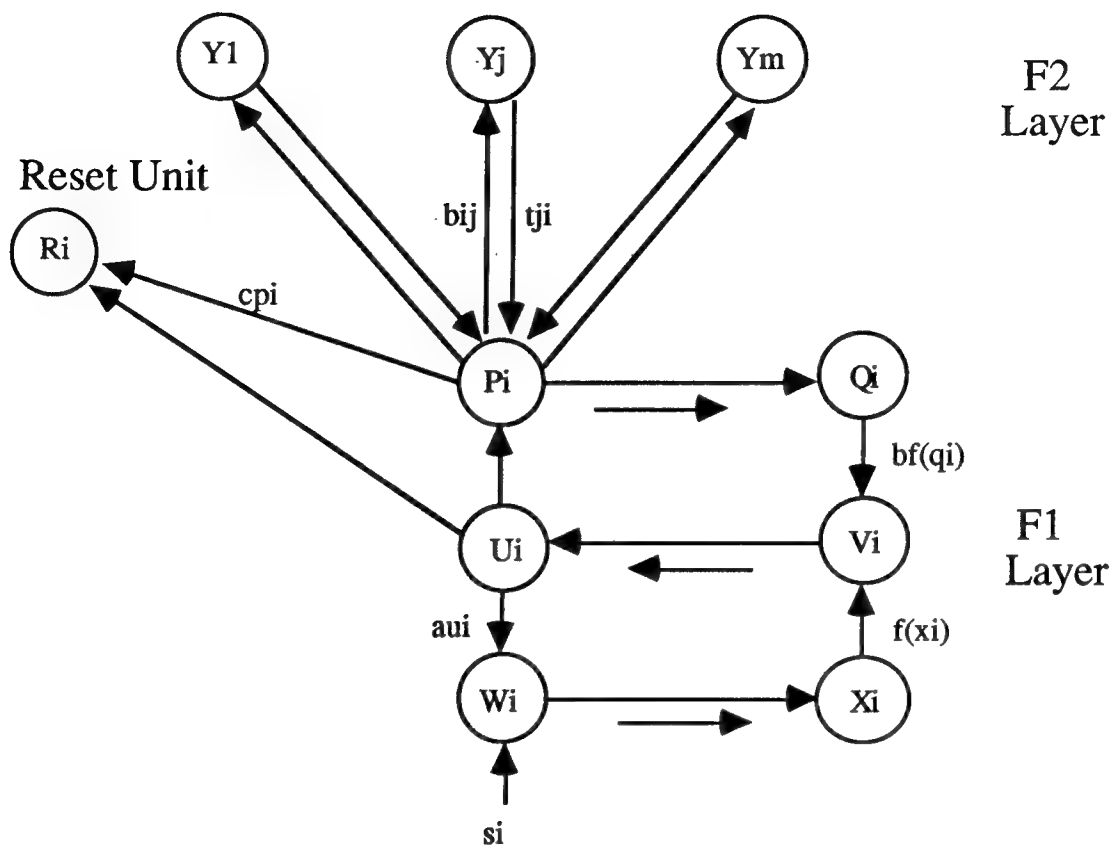
Figure 5.6. Typical ART2 Architecture.

3. Second cycle of updating the F1 layer.

$$u_i = \frac{v_i}{\|v\|} \qquad x_i = \frac{w_i}{\|w\|}$$

$$w_i = s_i + au_i \qquad q_i = \frac{p_i}{\|p\|}$$

$$p_i = u_i \qquad v_i = f(x_i) + bf(q_i)$$

4. The P units send their signals to the F2 layer, where a winner-take-all competition chooses the candidate cluster unit to learn the input pattern. The winner sets its activation to d, 0<d<1.

$$y_j = \sum_i b_{ij} p_i$$

5. The units Ui and Pi send signals to the corresponding reset unit. If the reset is false, do slow learning.

Check for reset :

$$u_i = \frac{v_i}{\|v\|} \qquad p_i = u_i + dt_{ji}$$

$$r_i = \frac{u_i + cp_i}{\|u\| + c\|p\|}$$

if $\|r\| \geq \rho$, where ρ is vigilance parameter, reset is false, do slow learning

$$w_i = s_i + au_i$$

$$x_i = \frac{w_i}{\|w\|}$$

$$q_i = \frac{p_i}{\|p\|}$$

$$v_i = f(x_i) + bf(q_i)$$

$$t_{ji} = \alpha du_i + \{1 + \alpha d(d-1)\}t_{ji}$$

$$b_{ij} = \alpha du_i + \{1 + \alpha d(d-1)\}b_{ij}, \quad \text{where } \alpha \text{ is learning rate}$$

6. Otherwise, the candidate unit is rejected. Choose another candidate and go back to 5.

The process continues until either a satisfactory match is found or all units are inhibited, which means it can't be clustered into any clusters. We put such paths into CNC.

5.4.2. Preprocessing of Source Simulation Data.

All source data are generated by running COSAGE on SPARC station. Seed 1 to 18 are used to generate data file p16.1 -- p16.18 and p55.1 -- p55.18. At current study, p16.1 -- p16.18 are chosen as sample data.

Each data file consists of 12 parts, representing the simulation result of every 4 hours in total 48 hours. Currently three of them (4th, 8th and 12th) are used which are the simulation result of 16th, 32nd and 48th hours. Then we compute the number of kills in these three time intervals. (0--16, 16--32 and 32--48).

Select 15 items according to the "firing equipment, munition name, target name or mission type" pair. (see **Table 5.1**). There are generally more than 300 items. However most of them are 0 or small numbers. So we choose 15 items which we think can represent the characteristic of the simulation result. Compute the average number of kills for each item and time interval over all 18 simulation paths, the number of kills of different item at different time interval will be "1" if it is bigger than its corresponding average value. Otherwise it will be "0". Because each path has 3

time intervals and 15 items, so now each path can be represented by a 45 dimension binary vector, which can be handled by ART2 neural network.

Table 5.1. Firing Equipment, Munition and Target Pair.		
Firing Equipment	Munition Name	Target Name
UH155Z	M795	RFATP
UH155Z	M795	RINTP
UH155Z	M795	RM-TP
UH155Z	M483A1	RFATP
UH155Z	M483A1	RTRUCK
UH155Z	M483A1	RINTP
UH155A	M795	RFATP
UH155A	M483A1	RFATP
UH203Z	M509A1	RFATP
UH203Z	M509A1	RTRUCK
RH152Z	152ICM	UFATP
RH152Z	H152HF	UFATP
UMLRS	MLRS	RFATP
UMLRS	MLRS	RTRUCK
UH105A	M1	RM-TP

5.4.3. Results.

For the results that follow,

cluster 1:   path 5, path 6

means that sample paths 5 and 6 are clustered into the same cluster, in this case, cluster 1.

CNC :   path 10

means that sample path 10 cannot be clustered into any of the other clusters. The vigilance parameter controls the degree of similarity that sample paths must have in order to be placed into the same cluster. A high vigilance parameter means that the sample paths must be virtually identical before they will be placed into the same cluster, and a low vigilance parameter is more tolerant of differences.

Example 1:

vigilance parameter: 0.99

cluster 1:   path 7

cluster 2:   path 2

cluster 3:   path 3

cluster 4:   path 6

cluster 5:   path 10

cluster 6:   path 14

cluster 7:   path 1

cluster 8:   path 11

cluster 9:   path 15

cluster 10:   path 4

CNC: path 5, path 8, path 9, path 12, path 13, path 16, path 17, path 18

Analysis: Here the vigilance parameter is too big. So there are many sample paths which cannot be clustered.

Example 2:

vigilance parameter: 0.95

cluster 1:   path 1, path 7, path 15, path 17

cluster 2:   path 2, path 8, path 11

cluster 3:   path 3, path 5, path 6 , path 12 , path 14

cluster 4:   path 10, path 13, path 16

cluster 5:   path 9

cluster 6:   path 4

cluster 7:   path 18

cluster 8:

cluster 9:

cluster 10:

CNC:

Analysis: The vigilance parameter is still too big.

Example 3:

vigilance parameter: 0.90

cluster 1:   path 1 , path 7 , path 15 , path 17 , path 18

cluster 2:   path 2 , path 8 , path 11

cluster 3:   path 3 , path 4 , path 5 , path 6 , path 9 , path 12 , path 14

cluster 4:   path 10 , path 13 , path 16

cluster 5:

cluster 6:

cluster 7:

cluster 8:

cluster 9:

cluster 10:

CNC:

Analysis: A vigilance parameter value of 0.90 appears to be a good one.

Example 4:

vigilance parameter: 0.85

cluster 1:   path 1 , path 7 , path 15 , path 17 , path 18

cluster 2:   path 2 , path 8 , path 11

cluster 3:   path 3 , path 4 , path 5 , path 6 , path 9 , path 12 , path 14

cluster 4:   path 10 , path 13 , path 16

cluster 5:

cluster 6:

cluster 7:

cluster 8:

cluster 9:

cluster 10:

CNC:

Analysis: We will choose a vigilance parameter of 0.90.

Sensitivity to The Input Order:

For the previous examples, the input ordering of the sample paths was sequential from 1 to 18. Here we show that although the resulting clustering varies with the input order, the variation is acceptable.

Example 1:

input order: 1 to 18

vigilance parameter: 0.90

cluster 1:   path 1 , path 7 , path 15 , path 17 , path 18

cluster 2:   path 2 , path 8 , path 11

cluster 3:   *path 3, path 4, path 5, path 6, path 9, path12,path 14*

cluster 4:   path 10 , path 13 , path 16

cluster 5:

cluster 6:
cluster 7:
cluster 8:
cluster 9:
cluster 10:
CNC:

Example 2:

input order: 1 8 15 2 9 16 3 10 17 4 11 18 5 12 6 13 7 14
vigilance parameter: 0.90
cluster 1:   path 1 , path 7 ,path 17 , path 10 , path 13 , path 16
cluster 2:   path 2 , path 8 , path 11 , *path 3 , path 4 , path 14* , path 15 , path 18
cluster 3:   *path 5 , path 6 , path 9 , path 12*
cluster 4:
cluster 5:
cluster 6:
cluster 7:
cluster 8:
cluster 9:
cluster 10:
CNC:

Example 3:

input order: 2 3 4 5 6 7 17 8 9 10 13 12 14 1 15 16 11 18
vigilance parameter: 0.90
cluster 1:   path 2 , path 8 , path 11
cluster 2:   *path 3 , path 4 , path 5 , path 6 , path 9 , path 12 , path 14*
cluster 3:   path 1 , path 7 , path 15 , path 17 , path 18
cluster 4:   path 10 , path 13 , path 16
cluster 5:
 cluster 6:
cluster 7:
cluster 8:
cluster 9:
cluster 10:
CNC:

Example 4:

input order: 10 13 5 6 12 14 1 15 2 3 16 11 18 4 7 17 8 9

vigilance parameter: 0.90

cluster 1:   path 10 , path 13 , path 16

cluster 2:   path 2, path 8, path 11 ,*path 3, path 4, path 5, path 6 , path9, path 12, path 14*

cluster 3:   path 1

cluster 4:   path 7 , path 15 , path 17 , path18

cluster 5:

cluster 6:

cluster 7:

cluster 8:

cluster 9:

cluster 10:

CNC:

Example 5:

input order: 1 13 5 12 3 4 9 10 11 14 6 18 8 17 7 16 15 2

vigilance parameter: 0.90

cluster 1:   path 1 , path 7 , path 18

cluster 2:   path 10 , path 13 , path 16

cluster 3:   path 2 , *path 3 , path 4 , path 5 , path 6 , path 9 , path 12 , path 14*

cluster 4:   path 8 , path 11 , path 15 , path 17

cluster 5:

cluster 6:

cluster 7:

cluster 8:

cluster 9:

cluster 10:

CNC:

Example 6:

input order: 7 13 14 8 5 15 18 3 11 10 16 4 17 6 12 1 2 9

vigilance parameter: 0.90

cluster 1:   path 1 , path 7 , path 15 , path17 , path 18 , *path 6*

cluster 2:   path 2 , path 8 , path 11 ,*path 3 , path 4 , path 5 , path 9 , path 12 , path 14 ,*
                path 10 , path 13 , path16

cluster 3:

cluster 4:

cluster 5:

cluster 6:

cluster 7:

cluster 8:

cluster 9:

cluster 10:

CNC:

Analysis: In comparing examples 1 through 6 above, we see that although the clusters vary with the input order, such variation is tolerable.

5.5. Summary.

Through this project we have gained a thorough understanding of the importance of preserving the stochastic fidelity in hierarchical battle simulation models. The Lanchester equation and similar equations can only be used for average values and has little value in preserving the stochastic fidelity. Ito type of stochastic equations can be used to enhance the deterministic Lanchester equation. However the computational complexity involved could be insurmountable. Path bundle grouping approach seems to be computationally feasible and bears strong "physical" meaning to render it easy to implement in practice. On the other hand, since we will have to deal with random vectors with huge dimension, neural networks seem to be the only feasible computation tool for this purpose. Preliminary experiments with COSAGE output show that this approach indeed works and we propose to pursue this direction vigorously in the next period.

6. THE AIRCRAFT REFUELING AND MAINTENANCE SYSTEM.

While TERSM has provided an excellent testbed to study the feasibility of the CCNN metamodeling approach, it lacks some of the features that constitute real challenges to metamodeling. For one, TERSM lacks significant randomness. As provided to us, TERSM is deterministic. However, by a simple modification to the TERSM program files, it can be made stochastic. Once this modification has been made, the number of emitters detected will change as a function of the initial random number seed. The change in the number of emitters detected, however, is insignificant and the simulator is, for all intents and purposes, deterministic. For another, TERSM does not exhibit asymptotic relationships which are very common in practice. Exposing such asymptotic behaviors often requires very long simulation runs. Avoiding long simulation runs is one of the real benefits of metamodeling. TERSM, on the other hand, has a running time of just under a minute on a modern workstation. For such a simulation, the benefits of metamodeling are not so clear. Additionally, asymptotic behaviors can be very difficult for polynomials to capture, and provide one of the primary motivations for the use of more powerful metamodeling methods like the CCNN we are investigating.

We have therefore concentrated on identifying a good benchmark problem with the above features. In this effort, we have consulted a number of references from the literature of military models [20]-[22]. In [22], for example, a model is considered for the process of moving and reassigning strategic airlift pilots with the objective of managing and ultimately minimizing moving costs while maintaining mission capability. Motivated by this problem, we have created a model which may be used to represent several interesting Air Force C^3I operations. Thus far we have restricted our investigations to a version we call the *Aircraft Refueling and Maintenance System* (ARMS). This system can be viewed as a high-resolution component of a combat simulation model whose output is used by a lower-resolution model. Thus, issues related to hierarchical decomposition can also be studied through the ARMS model.

In this section we describe the ARMS model and present some preliminary results to demonstrate its features and complexity. A definition of the ARMS model is presented in Section 6.1. Section 6.2 contains the results of a detailed simulation study of the model. Remarks about the model are contained in Section 6.3.

6.1. Problem Formulation.

The basic ARMS model is shown in **Fig. 6.1**. As illustrated, ARMS is a multiclass queueing system. Jobs from each class $n = 1,...,N$ arrive with average rates λ_n to separate arrival queues with capacities C_n (possibly infinite). The system has θ tokens. At block 1, the jobs compete for tokens on a priority basis, with the class 1 jobs having the highest priority. The job waiting in the

highest priority arrival queue will be the first to get a token when one becomes available. After getting a token, the jobs enter a service queue with capacity C_s. At block 2 the jobs are selected from the service queue according to some service discipline (e.g., first in first out (FIFO)) and routed to one of $k = 1,...,K$ servers. The time to service a job is a random variable $\mu(n,k)$ which can vary as a function of the job class n and the particular server k. Upon completing service, the jobs proceed to block 3, where the token is returned to a token pool, and the job leaves the system.

λ_1 →

λ_2 →

λ_N →

C_n

tokens θ

block 1

C_s

block 2

1
2
K
servers

block 3
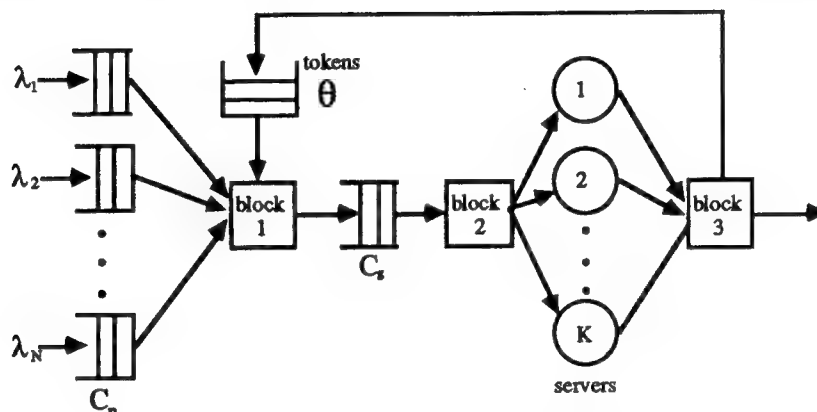
Figure 6.1. The basic ARMS queueing model.

The basic ARMS queueing model is very general and can be used to represent a large variety of Air Force C³I operations. For example, such a model can be used to represent computer networks, communications systems, or logistics problems. The specific problem we will consider is the aircraft refueling and maintenance system (ARMS) shown in **Fig. 6.2**. The ARMS problem has aircraft requesting to land at a particular site (airport or aircraft carrier) for refueling and/or maintenance purposes. Depending on aircraft type, a priority is assigned to each aircraft so that high-priority ones are served first. Since landing capacity and associated maintenance resources are limited, a specific number of "permits" (i.e., tokens) are available. An aircraft is, therefore, forced to wait until it receives a permit. Upon receiving a permit, the aircraft is guided to a refueling/maintenance area. If the resources required to complete the refueling/maintenance process are not immediately available (e.g., personnel, tools, spare parts, fuel), the aircraft is further delayed. When the aircraft completes service, the permit is returned to the permit pool, and the aircraft proceeds to take off and return to action. In studying this system, one is interested in minimizing the expected "down time" of an aircraft, with more emphasis given to certain types of aircraft (the ones given higher priority). At the same time, one is interested in keeping service costs within acceptable levels. From a modeling standpoint, one must therefore determine functional relationships such as the expected down time of a priority 1 aircraft with respect to factors such as the number of permits; or the number of maintenance resources allocated to the refueling/maintenance process.
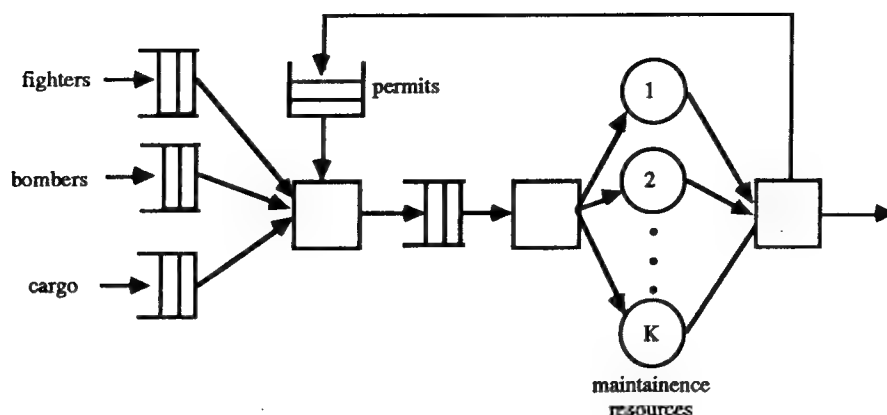
Figure 6.2. The aircraft refueling and maintenance system (ARMS).

6.2. Preliminary Analysis.

Our preliminary analysis focused on the 3-class, single server ARMS model shown in **Fig. 6.3**. In this model, the class 1 jobs have the highest priority, and the class 3 jobs the lowest. Job arrivals are assumed Poisson with rates λ_n, where n is the customer class. In order for a job to be served, it must have one of θ tokens. Jobs with tokens queue up to be served by a single server. At the completion of service, the jobs leave the system, and the tokens are returned to the token pool for reuse. When different classes of jobs are competing for tokens, the class with the highest priority gets one first. Jobs in the same class compete for tokens on a first come, first served (FCFS) basis. The arrival queues have capacity C_n, and the server queue has capacity $C_s = \theta$. Jobs in the server queue may be served FIFO (with no distinction made between jobs from different classes), or they may be served according to priority (with the highest priority jobs moving to the front of the queue). In any case, service is nonpremptive (once a job begins service, service cannot be interrupted, and will continue until completion), and the service time is an exponential random variable with parameter μ_n.
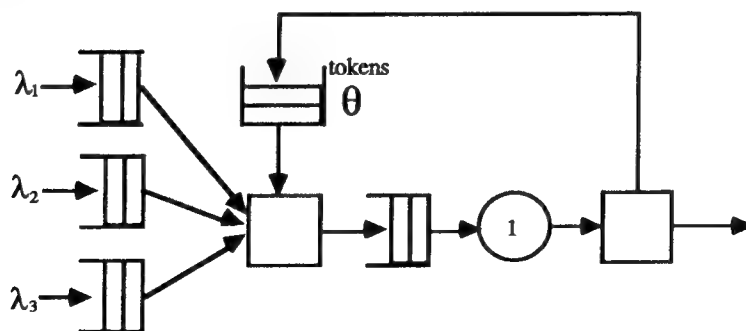
Figure 6.3. System used for analysis.

6.2.1. Discrete Event Model.

Next we present a detailed discrete event system (DES) description to explain the workings of the ARMS model in **Fig. 6.3**. To do so, we define the state of the model as follows,

$Q_n \in 0,1,...,C_n$ — number of class n jobs in the n-th arrival queue

$Q_\theta \in 0,1,...,\theta$ — number of tokens in the token pool

$Q_s \in 0,1,...,\theta$ — number of jobs in the server queue (recall $C_s = \theta$)

where $n \in 1,2,3$. State changes in the system are caused by four types of events,

A_n — arrival of a class n job to the n-th arrival queue

A_θ — arrival of a token to the token pool

A_s — arrival of a job+token pair to the server queue

D_s — departure of a job+token pair from the server

Job arrival events are always feasible. Token arrival events are only feasible when there are jobs being serviced (i.e., when there are jobs in possession of a token). Server arrival events are only feasible when tokens are available in the token pool. Departure events from the server are only feasible when there a job is being serviced.

  The state transition mechanism describing how the state of the system changes in response to the various events is given below.

• Job Arrival – A_n

   (1) $Q_n > 0, Q_n < C_n \Rightarrow Q_n = Q_n + 1$

   (2) $Q_n = C_n \Rightarrow Q_n = Q_n$, record blocking of a class n job

   (3) $Q_n = 0, Q_\theta = 0 \Rightarrow Q_n = 1$

   (4) $Q_n = 0, Q_\theta > 0 \Rightarrow Q_\theta = Q_\theta - 1$, schedule A_s to occur immediately

   (5) $Q_n = 0, Q_\theta > 0, Q_s = 0 \Rightarrow Q_\theta = Q_\theta - 1$, schedule D_s to occur in μ_n seconds

In equation (1) a class n job arrives to find its arrival queue not empty, but not at capacity either. The job is added to the back of the queue, where it must wait behind the other jobs in the queue before it can compete for a token. Jobs in the same class compete FCFS with other jobs in the same class for a token. In equation (2) a class n job arrives to find its arrival queue at capacity. Since the arrival queue is full, the job is blocked, and not allowed to enter the system. Blocking is usually undesirable, and should generally be avoided. In equation (3) a class n job arrives to an empty arrival queue, but finds that no tokens are available. The job, therefore, must wait in its arrival queue for a token to become available. In equation (4) a class n job arrives to an empty

arrival queue, and finds a token available (this implies that all other arrival queues must be empty). The job takes the token from the token pool, and proceeds to the server queue. This triggers an A_s event to occur immediately. Finally in equation (5) a class n job arrives to find an empty arrival queue, an available token, and an empty server queue. This job takes a token from the token pool, and immediately begins service. The service time for the job is μ_n, and a service completion event D_s is scheduled to take place in m$_n$ seconds.

- Service Completion – D_s

         (1) $Q_s = 0 \Rightarrow$ record performance for this job, schedule A_θ

         (2) $Q_s > 0 \Rightarrow$ record performance for this job, schedule A_θ, $Q_s = Q_s - 1$, schedule D_s

In equation (1) a job has just completed service and no other jobs are waiting in the server queue for service. The system time (down time) for the job is recorded, the job leaves the system, and the token is returned to the token pool (by scheduling an A_θ event to occur immediately). In equation (2) a job has just completed service, and other jobs are waiting for service in the server queue. As before, the system time for the job that just completed service is recorded, the job leaves the system, and the token is returned to the token pool. In addition, the server queue is decremented, and the job at the front of the server queue begins service. The time to service this job will be μ_n, and a service completion event D_s is scheduled to take place in μ_n seconds. Note, when the server queue is a priority queue, the high priority jobs are shuffled to the front of the queue. Otherwise, jobs are served in the order they arrived to the server queue, independent of their priority.

- Server Queue Arrival – A_s

         (1) $Q_s = 0 \Rightarrow$ schedule D_s

         (2) $Q_s > 0 \Rightarrow Q_s = Q_s + 1$

In equation (1) a job+token pair arrives to the server queue. Since the server queue is empty, the job immediately begins service. The time to service the job is μ_n, and a service completion event D_s is scheduled to take place in μ_n seconds. In equation (2) a job+token pair arrives to the server queue. In this case, the server queue is not empty, so the job is added to the queue. If the server queue is a priority queue, the job is placed behind the last job in its class. Otherwise, the job is placed at the back of the queue, independent of its priority.

- Token Queue Arrival – A_θ

$$(1) \quad Q_\theta > 0 \Rightarrow Q_\theta = Q_\theta + 1$$
$$(2) \quad Q_\theta = 0, Q_n > 0 \Rightarrow Q_n = Q_n - 1 \text{ (highest priority), schedule } A_s$$
$$(3) \quad Q_\theta = 0, Q_n = 0 \Rightarrow Q_\theta = 1$$

In equation (1) a token is returned to the token pool. Since the token pool is not empty, the token is added to the pool. Note, the token pool will contain tokens only when all arrival queues are empty. In equation (2) a token returning to the token pool finds that the token pool is empty, and that there is a job in at least one of the arrival queues. In this case, the job at the front of the highest priority arrival queue takes the token, leaves its arrival queue, and proceeds to the server queue. In equation (3) a token returning to the token pool finds that the token pool is empty, and that there are no jobs waiting in any of the arrival queues. In this case, the token remains in the token pool.

Based on the DES description above, a simulator was developed, and data were collected to assess the effects that the various parameters have on the system performance. The performance measure we use is the *mean system time* for each job class. The system time is the interval from when a job arrives to the system to the time the job completes service and leaves the system. In the ARMS problem, the service time is the "down time" of an aircraft. For the experiments that follow, the arrival queues are assumed to have infinite capacities, i.e., $C_n = \infty$ for $n = 1,2,3$.

6.2.2. Preliminary Arrival Rate Analysis.

Here we examine how changing the arrival rate of the class n jobs effects the system times of the other classes of jobs. To isolate the effects of changes in arrival rates, we set the number of tokens to a large value $\theta = 20$. When the number of tokens is large, the token loop can be neglected, and the system behaves as if the token loop is not there. The service times for each job class is set to $\mu_1 = \mu_2 = \mu_3 = 1$ second. We varied the arrival rate for one class at a time. The arrival rates for the classes we were not varying were set to $\lambda = 0.1$ job/second.

Figure 6.4 shows the effect of changing the arrival rate of the class 1 jobs. In this case, jobs in the server queue are served FIFO, with no distinction made between classes. As can be seen, increasing the arrival rate causes not only increases the system time of the class 1 jobs, but it also increases the system times of the other two classes. This happens because as the arrival rate of class 1 jobs increases, a queue builds in the class 1 arrival queue. Then whenever a token becomes available, a class 1 job will take it. The class 2 jobs will only get a token when the class 1 arrival queue is empty, and the class 3 jobs will only get a token when *both* the class 1 and class 2 arrival queues are empty. Notice the asymptotic behavior in the system time as the arrival rate increases.

Had we plotted the arrival rate for increasingly higher arrival rates, we would have seen the following behavior: First the system time of the class 3 jobs would go to infinity as queues build in the class 1 and class 2 arrival queues, and the class 3 jobs never get a token. As the arrival rate is increased still further, the system time of the class 2 jobs would go to infinity as the class 1 queue builds. Finally, at even higher arrival rates, the service time of the class 1 jobs would go to infinity when their arrival rate exceeds the rate at which the system can process them.
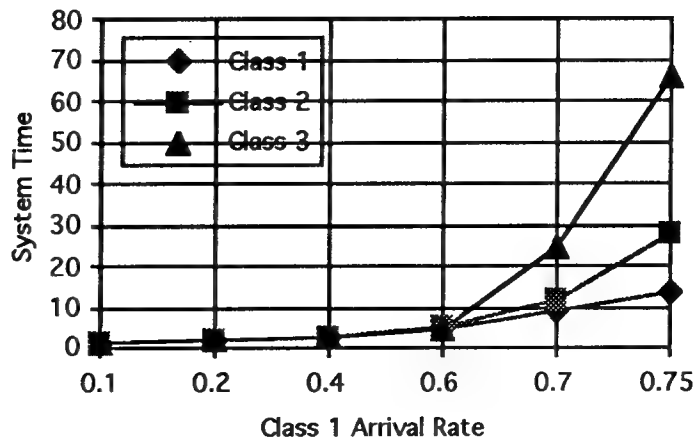
Figure 6.4. Effect of changing the class 1 arrival rate when the server queue is FIFO.

Figure 6.5 shows the effect of changing the arrival rate of the class 2 jobs. For **Fig. 6.5,** the server queue is FIFO, and there is no distinction made between classes. Here, increasing the service time of the class 2 jobs causes an increase in the system time of the class 1 jobs by forcing them to wait in the server queue for the class 2 jobs to be served. If we were to continue to increase the arrival rate of the class 2 jobs we would see the following behavior. First, as class 2 jobs began to accumulate in the class 2 arrival queue, they would starve the class 3 jobs of tokens, and the system time of the class 3 jobs would go to infinity. As we increase the arrival rate of the class 2 jobs still further, they would begin to arrive faster than the system can process them, and the system time of the class 2 jobs would go to infinity. Notice, however, that because there are only $\theta = 20$ tokens, the system time of the class 1 jobs will not go much above 21 seconds, regardless of how high the arrival rate of the class 2 jobs becomes. To see this, suppose that all of the tokens are in use. An arriving class 1 job will, therefore, have to wait about 1 second for a token to become available. Then, regardless of how many jobs are in the class 2 arrival queue, the class 1 job will take the token. The job+token pair will proceed to the server queue, where it will have to wait for the 19 jobs in front of it to complete service. These jobs take an average of 1 second to complete service, and service on the job itself takes another second. Thus, the service time for the class 1 job will be $\approx 1+19+1 = 21$ seconds.
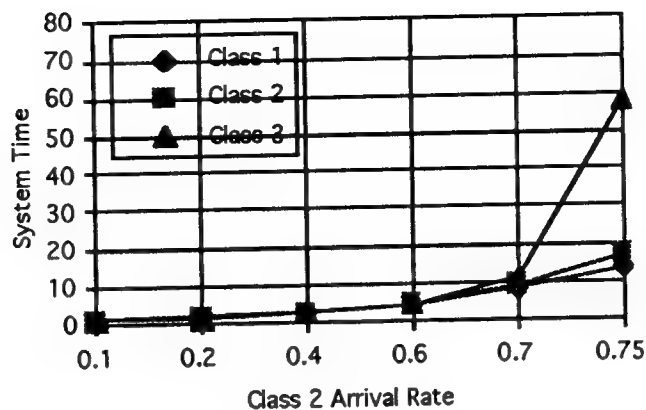
Figure 6.5. Effect of changing the class 2 arrival rate when the server queue is FIFO.

There are numerous additional issues to pursue regarding the behavior of this system. Once these issues are understood, then the basic input-output relationships required to define a metamodel can be identified and the techniques discussed in Sections 2-3 of this report can be applied. We leave this as the topic for our future work and only summarize below some of the issues we have identified to date: the effect of introducing priorities to the server queue in our model; the effect of changing the third class's arrival rate; the effect of varying service times in this model (i.e., providing faster processing of aircraft); and the effect of the number of tokens in this model (i.e., adding or eliminating personnel in the ARMS). Moreover, we have limited ourselves so far to the mean system time as a performance measure of interest. Clearly, there are several additional measures to be considered, including the capacities of the various queues which have so far been assumed to be infinite.

6.3. Application of Concurrent Simulation to ARMS

One of our objectives in using the ARMS benchmark problem is to use concurrent simulation to collect the data that will be needed to identify a metamodel for ARMS. Up to this point, all of our concurrent simulation experiments have been done using custom simulation programs written in the C programming language. This has meant that for every new system we wanted to simulate, we have had to write a new program from scratch. To avoid this wasteful effort, we have initiated an effort for developing an object oriented discrete event system simulation environment. The objects in this environment are the building blocks of discrete event systems: queues, servers, and so forth. By writing simple code that describes how the objects are connected together we can simulate different systems. Because the objects are error-free and reusable, this greatly speeds up the code writing and debugging phase, allowing us to concentrate on the algorithms instead of the code. The following gives a brief description of our efforts to date.

6.3.1. Object Oriented Concurrent Simulator (OOCS)

At this point, the simulator consists of four classes/structures of objects:

1. A class that describes the random number generator (RNG).
2. A class that describes each type of building block: queues, servers etc.
3. A structure that describes each "entity" (customer) in the system.
4. A structure that describes each possible event.

The RNG is the means by which we introduce randomness into the simulation model. Its purpose is to supply the event lifetimes. Each building block has a set of variables that define the topology of the system being simulated. These variables keep information about how the blocks are connected together, information about the state of each block (e.g., the number of entities currently in the block), and a unique sequence number to identify it. In addition, each entity has a set of variables to describe its attributes (e.g., priority). Associated with each building block are four types of events:

1. *Arrival* event which occurs when an entity arrives at the block.
2. *Departure* event which occurs when an entity leaves the block.
3. *Completion* event which occurs when a "server" block finishes serving an entity.
4. *Message* event which is used to communicate information from one block to the other.

The following four event types drive the system:

- A *Departure* event from a block triggers an *Arrival* event at the next block downstream. It will also trigger a *Departure* event to itself if it has more entities to release (e.g., a queue with waiting entities), and to the previous block upstream if that block was stalled because of blockage downstream.
- An *Arrival* event from a block triggers either a *Completion* event or a *Departure* event to the same block.
- A *Completion* event from a block triggers either a *Departure* event to the same block (communication blocking) or a *Message* event to a downstream block asking whether there is buffering space to release the entity (manufacturing blocking).
- A *Message* event from a block either triggers a *Departure* event from the upstream block or puts the upstream block in a blocked state.

Each block has four functions to determine how the block will respond to each of the four types of events. Note that some blocks do not do anything for certain types of events in which case the corresponding function is null. For example, a *Completion* event does not cause any action from a queue. So far, the following blocks are implemented:

- *Generator;*
  Generates entities that enter the simulation model.
- *Server;*

Takes an entity, provides some service, and then releases the entity.

- *BatchServer;*

    Similar to the server above, however, it only starts service when all required parts are available. This block has several input connectors, when all connectors have an entity present, it takes all of them "batches" them into a single entity, provides some service and then releases them as one entity.

- *UnBatchServer;*

    This is the opposite of the BatchServer. This block has several output connectors. It takes an entity which has been "batched", provides some service and then breaks the batch and releases one entity to each of its output connectors.

- *FIFOQ;* (FIFO queue)

    This is a First In First Out Queue.

- *PriorityQ;* (Priority Queue)

    A queue that looks into the priority field of each entity and places them in the queue appropriately.

- *P_Router;* (Probabilistic router)

    Takes an entity in and routes it to one of its outputs based on some probability.

- *Merge;*

    Connects several output connectors to a single input connector.

- Exit;

    The entity is out of the scope of the model, so release the memory that the entity holds.

With the blocks described above, we are able to simulate virtually any queueing model, including ARMS.

6.3.2. Simulation with Recycled Event Lifetimes

In concurrent simulation (see Section 2), we saw that there are some coupling dynamics between the nominal and perturbed sample paths which are necessary in order to determine whether the perturbed sample path is active or not. Therefore, it is reasonable to believe that the implementation of the coupling induces some additional overhead. It turns out that if we relax the requirement of generating performance estimates under several parameters *simultaneously*, we can still save some time and improve the performance of the concurrent simulation. In other words, if we don't care for performance estimates *while* simulation is running, but only in the performance after all simulations are done, then one can use what we call *Simulation with Recycled Event Lifetimes* in order to reduce the total simulation time even more.

Simulation with Recycled Event Lifetimes is based on the observation that concurrent simulation can reduce the total simulation time because it reduces the total number of random

variates generated by a random number generator, which in general is a time consuming process. In this approach, we simulate the nominal sample path like we normally do in brute force simulation but rather that discarding all the random variates that constitute the event lifetimes we save them in the memory. When we simulate all other sample paths we do exactly as we do in brute force simulation but rather than generating a new random variate we use one that has been stored in the memory during the simulation of the nominal sample path. In this way we trade off computer memory for higher speedup.

6.3.3. Results using the OOCS

We have implemented several queueing systems using the OOCS to test the feasibility of the approach. **Table 6.1** compares the speedup (over brute force simulation) between concurrent simulation and brute force simulation with recycled lifetimes for the following systems:

- **Single Server Queueing System:** This is a system that consists of a single server fed by a finite capacity queue. Jobs arrive and receive service on a First Come First Serve basis. The parameter of interest is the queue capacity and the performance measure is the blocking probability.

- *ARMS*: This is the system described in the previous sections. The parameter of interest is the number of tokens and the performance measure is the system time of each job.

These results clearly indicate that the computational overhead due to an object-oriented approach reduces the efficiency of concurrent simulation. This is not surprising: a substantial overhead created by object-oriented structures in simulation is something that is commonly observed in commercial packages that have adopted such structures. As our results indicate, we can compensate for this fact through the recycling approach described above. We should point out, however, that these results are preliminary and that opportunities for further improvements in the implementation of the TWA (see section 2.3.1) have yet to be explored.

System		Speedup
M/M/1/K Exponential Distributions	Concurrent Simulation Recycled Lifetimes	0.940 1.111
G/G/1/K Erlang Distributions- order 30	Concurrent Simulation Recycled Lifetimes	1.633 1.783
ARMS Exponential Distributions	Concurrent Simulation Recycled Lifetimes	0.951 1.081
ARMS Erlang Distributions-order 30	Concurrent Simulation Recycled Lifetimes	1.590 1.686

Table 6.1: Speedup factors using OOCS

6.4. Conclusions.

Although the basic ARMS model is fairly simple to describe, it admits a great deal of complexity, and possesses many features which constitute real challenges to metamodeling. Three aspects of the model that make it especially challenging: Multiple customer classes; the possibility of blocking when the queues have finite capacities; and simultaneous resource possession (a job must have a token). Systems with these properties are notoriously difficult to analyze because they lack convenient product form solutions. For such systems, simulation provides the only practical way to obtain performance results.

Simulating such systems can also pose challenges. As is typical of systems involving queues, they can become unstable when the arrival rate gets too high or the service time gets too long. This instability is characterized by the unbounded growth in queue lengths. However, to ensure maximum utilization of resources, it is often desirable to operate near the point of instability. Unfortunately, simulation results obtained when a system is operating near the point of instability are often statistically unreliable unless the simulation is run for a very long time.

Since there are many parameters which can be adjusted, the basic ARMS model is very complex. As a part of a battle simulation, one would be interested in answering "what if" questions. For example, one might want to know the optimal parameter settings to minimize the aircraft "down time." Answering this question involves determining the combined effect of variation in such things as: the number of tokens; the queueing discipline in the server queue; the number of servers; and the routing policy from the server queue to the servers.

The features of the basic ARMS model—simple description, asymptotic behaviors, multiple classes, blocking, simultaneous resource possession, and routing—make it an excellent benchmark problem for metamodeling studies. In future work we will develop metamodels for the basic ARMS model using the CCNN. Training the CCNN, however, will require many training runs under many different parameter settings. By exploiting the concurrent estimation techniques described in Section 2, we hope to develop strategies for quickly and efficiently collecting the training data needed for metamodel construction.

7. CONCLUSIONS AND FUTURE RESEARCH DIRECTIONS.

In this section, we summarize the main findings, lessons learned, and recommendations for future directions that have resulted from this project.

• **Concurrent and Parallel Simulation.** Computer simulation has emerged as the only tool of universal applicability when it comes to modeling complex systems. However, systematic design and performance studies of such systems require a large number of time-consuming simulation runs: to ask N "what if" questions, a nominal simulation run is needed, followed by N additional runs, one for each "what if". One of the main goals of this project has been to establish the feasibility of the idea that multiple "what if" questions can be answered from data obtained from *the nominal simulation run alone*. This approach has been termed "concurrent simulation" when applied to conventional sequential computers, in contrast to "parallel simulation" when applied to parallel computers or networks of workstations. A basic theoretical framework underlying the concurrent simulation algorithms is one of the accomplishments of this project. Based on this framework, we have developed a general (i.e., *not limited* by specific modeling assumptions) concurrent simulation approach implemented through a *Time Warping Algorithm* (TWA). To quantify and compare the effectiveness of this algorithm to conventional repetitive simulation, we have also introduced the concept of "speedup". Using a "speedup factor" metric, a number of detailed numerical examples were provided in this report to demonstrate the effectiveness of that TWA.

We summarize next some lessons we have learned from this work, which also suggest directions for future research:

(1) Implementation of concurrent simulation schemes is critical. We have found that the speedups achieved by concurrent simulation are significantly affected by such factors as the particular computer hardware used and the data structures selected. This suggests the need for a further exploration of different ways to implement the TWA and its variants.

(2) Parallel simulation (as defined above) promises at least one additional order of magnitude in speedup over concurrent simulation. In this project, we were limited to the use of sequential computers. It is clear, however, that when the state update component of a simulation process (the one we cannot "parallelize" through concurrent simulation) becomes large there is an opportunity to capitalize on the use of parallel processors. The integration of computer simulation methodologies with parallel processing architectures is a direction that holds great promise.

(3) Our speedup analysis in Section 2.4 has clearly shown that the benefit of concurrent simulation drastically increases with the stochastic complexity of the system being simulated. Since a considerable fraction of simulation time is devoted to the generation of random variates

for modeling stochastic features of a system, the more complex this generation mechanism is, the greater the speedup effect is, since concurrent simulation involves this mechanism only once over a large number of replications.

• **Metamodeling through Neural Networks.** The metamodeling procedure we have studied combines simulation of a complex system with the process of training a neural network to become a surrogate model of this system. This exploits the ability of a neural network to act as a universal function approximator. The challenge, however, is to identify an appropriate neural network size and detailed architecture for this task. Using a *Cascade Correlation Neural Network* (CCNN), a multilayer neural network that builds itself while it learns, we were able to develop procedures for training that are both adequately fast and highly accurate. Our results were applied to the Tactical Electronic Reconnaissance Simulator (TERSM) benchmark problem successfully demonstrating that (a) Neural networks can be trained through simulation to generate accurate metamodels for complex input-output relationships where classical polynomial-based techniques fail, and (b) Neural network metamodels are significantly more accurate than their polynomial counterparts (see detailed results in Sections 4.3 and 4.4).

Following are some lessons we have learned from this work, giving rise to new directions for future research:

(1) Determining the training data size for metamodeling through neural networks is an important issue. Since each input-output pair of data in this set is the result of a long simulation run, it is crucial to find answers to such questions as: what is the optimal training data size for a given problem? Can we use some kind of adaptive mechanism to find or approach this optimum? Can we segment the problem and develop separate models for different regions of the input space?

(2). It appears possible to improve learning efficiency and speed up the learning process of a neural network. In the course of our work, we were able to accomplish this through a variety of techniques. There are a number of possibilities that suggest themselves, but clearly require further investigation, such as: the possibility of introducing a non-linear activation function to the output unit of a CCNN; developing an adaptive mechanism for adjusting the learning rate that we can use to speedup the training; and starting from several different initial points (multi-starts) seems to substantially help training, but has yet to be studied in a systematic framework.

(3) Although the TERSM setting provided an excellent benchmark setting for our metamodeling studies, it soon became clear that more challenging benchmark models are needed. There are at least two features required for better testing purposes: (a) Models with substantial stochastic complexity, and (b) Functional relationships to be modeled which exhibit asymptotic

behavior, a very common occurrence in practice. It is for this reason that the latter part of this project became devoted to the development of a new benchmark problem. The ARMS model introduced in Section 6 is a step in that direction which we hope to further develop in the future.

• **Stochastic Fidelity in Hierarchical Simulation.** There are three important directions for future research. The first is to collect expert knowledge for help in resolving how to bundle high-resolution paths and when to invoke new killer/victim boards at the top level of the modeling hierarchy. Top level results should be consistent with not only means but higher moments of the underlying high-resolution distributions. The second direction is to permit division/brigade engagements of variable duration within the theater level simulation. Both directions are closely related to the stochastic fidelity issue, and we believe that the path bundle grouping approach is going to be very useful. The third direction is to develop fast clustering algorithms to group the huge dimensional data generated from the high-resolution simulator. We are investigating the use of ART2 neural network for this purpose.

• **Interactive Use of Simulation.** As pointed out earlier, the implementation of many of the techniques studied and developed in this project is an important and nontrivial task, greatly dependent on hardware and software technologies at one's disposal. Although our past work has demonstrated the interactive use of some of these techniques using a commercial simulation environment, this task was outside the scope of this project. We do wish to stress, however, that concurrent simulation and neural network technologies are ideally suited for emerging technologies, such as parallel processing, and new software capabilities that include Object Oriented Programming (OOP) and Graphical User Interfaces (GUI).

REFERENCES

[1]   Carpenter, G.A. and Stephen Grossberg, "ART 2: self-organization of stable category recognition codes for analog input patterns", *Applied Optics*, December 1987.

[2]   Cassandras, C.G., *Discrete Event Systems: Modeling and Performance Analysis.* Boston, MA: Richard D. Irwin, Inc., & Aksen Associates, Inc., 1993.

[3]   Cassandras, C.G. and J. Pan, "Parallel Sample Path Generation for Discrete Event Systems and the Traffic Smoothing Problem," *Journal of Discrete Event Dynamic Systems*, Vol. 5, No. 2/3, pp. 187-217, 1995.

[4]   Cassandras, C.G. and C. Panayiotou, "Concurrent Sample Path Estimation for Discrete Event Systems," to appear in the *Proceedings of the 35th Conference on Decision and Control*, 1996.

[5]   Cassandras, C.G. and S.G. Strickland, "On-Line Sensitivity Analysis of Markov Chains," *IEEE Trans. on Automatic Control*, Vol. AC-34(1): 76-86, 1989.

[6]   Cassandras, C.G. and S.G. Strickland, "Observable Augmented Systems for Sensitivity Analysis of Markov and Semi-Markov Processes," *IEEE Trans. on Automatic Control*, Vol. AC-34(10): 1026-1037, 1989.

[7]   Caughlin, D., "Verification, Validation, and Accreditation of Models and Simulations through Reduced Order Metamodels," in *Proceedings of the 1995 Winter Simulation Conference*, pp. 1405-1412, December 1995.

[8]   Cybenko, G. Approximation by superpositions of a sigmoid function. *Mathematics of Control, Signals, and Systems*, Vol. 2, pp. 304-314, 1989.

[9]   Darken, C., J. Chang, J. Moody, "Learning Rate Schedules for Faster Stochastic Gradient Search", Neural Networks for Signal Processing 2, *Proceedings of the 1992 IEEE Workshop on Neural Networks*.

[10]   Fahlman, S.E., and Lebiere, C., "The Cascade-Correlation Learning Architecture", Carnegie Mellon Univ., Technical Report CMU-CS-90-100, Feb., 1990.

[11]   Fausett, L., *Fundamentals of Neural Networks, Architecture, Algorithms and Applications*, Prentice Hall, 1994.

[12]   Glasserman, P., *Gradient Estimation Via Perturbation Analysis.* Boston, MA: Kluwer, 1991.

[13]   Hagiwara, M., "Theoretical Derivation of Momentum Term in Back-propagation", *1992 IEEE/INNS Conference*, pp. I682-I686.

[14]   Ho, Y.C., and X.R. Cao, *Perturbation Analysis of Discrete Event Dynamic Systems.* Kluwer Publishing, 1991.

[15] Hoehfeld, M., Fahlman, S.E., "Learning with Limited Numerical Precision Using the Cascade-Correlation Algorithm", *IEEE Trans. on Neural Networks*, Vol. 3., No. 4, July, 1992.

[16] Jacobs, R.A., "Increased Rates of Convergence Through Learning Rate Adaptation", *Neural Networks*, Vol. 1, pp. 295-307, 1988.

[17] Jain, A.K. and Dubes, R.C., *Algorithms for Clustering Data*, Prentice Hall, 1988.

[18] Kuan, C.M., K. Hornik, "Convergence of Learning Algorithms with Constant Learning Rates", *IEEE Transactions on Neural Networks*, Vol. 2, No. 5, Sept., 1991.

[19] Luger, G.F. and Stubblefield, W.A., *Artificial Intelligence, Structures and Strategies for Complex Problem Solving*, Benjamin/Cummings Publishing Co. 1993.

[20] Meidt, G. and Bauer, K.W. Jr., "PCRSM: A Decision Support System for Simulation Metamodel Construction", *Simulation*, Vol. 59, 3, pp. 183-191, 1992.

[21] Montgomery, D.C., *Design and Analysis of Experiments*, John-Wiley, 1984.

[22] Percich, D.M., "Modeling the Permanent Change of Station Moving Costs of Strategic Airlift Pilots", MS Thesis, Air Force Institute of Technology, 1987.

[23] Petersen, R.C., *Design and Analysis of Experiments*, Marcel Dekker, 1985.

[24] Phatak, D.S., Koren, I., "Connectivity and performance Tradeoffs in the Cascade correlation learning Architecture", Univ. of Mass, Technical Report No. TR-92-CSE-27, Aug., 1992.

[25] Robinson, S., "Scenario Analysis : What it is and how it works", Manuscript, University of Wisconsin at Madison, May 1990.

[26] Rummelhart, D.E. and J.L. McClelland (eds.). *Parallel Distributed Processing*, MIT Press, 1986.

[27] Shang, Y., B.W. Wah, "Global Optimization for Neural Network Training", *IEEE Computer*, March 1996.

[28] Tatum, F.A., "A Tactical Electronic Reconnaissance Simulation Model," *Rand Corporation Technical Report*, 24 October 1969.

[29] Vakili, P., "A Standard Clock Technique for Efficient Simulation," *Operations Research Letters*, Vol. 10, pp. 445-452, 1991.

[30] Weir, M.K., "A Method for Self-Determination of Adaptive Learning Rates in Back Propagation", Neural Networks, Vol. 4, pp. 371-379, 1991.

[31] Zeimer, M.A., and J.D. Tew, "Metamodel Applications Using TERSM," in *Proceedings of the 1995 Winter Simulation Conference*, pp. 1421-1428, December 1995.

[32] Zeimer, M.A., J.D. Tew, R.G. Sargent, and A. Sisti, "Metamodel Procedures for Air Engagement Simulation Models," *IRAE Technical Report,* Rome Laboratory, Griffis A.F.B., N.Y., January 1993.

APPENDIX

Table A.1

Xo	Yo	θ	V	Emitters with CEP < 5
400	300	288	1150	36
400	300	288	186	14
400	300	0	1150	0
400	300	0	186	0
400	0	288	1150	625
400	0	288	186	23
400	0	0	1150	194
400	0	0	186	9
0	300	288	1150	0
0	300	288	186	0
0	300	0	1150	0
0	300	0	186	0
0	0	288	1150	0
0	0	288	186	0
0	0	0	1150	148
0	0	0	186	2
200	150	144	668	446
0	150	144	668	341
400	150	144	668	16
200	0	144	668	0
200	300	144	668	636
200	150	0	668	335
200	150	288	668	474
200	150	144	186	472
200	150	144	1150	346
300	225	217.6	927.5	770
300	225	217.6	408.5	727
300	225	70.4	927.5	22
300	225	70.4	408.5	51
300	75	217.6	927.5	101
300	75	217.6	408.5	189
300	75	70.4	927.5	111
300	75	70.4	408.5	187
100	225	217.6	927.5	224
100	225	217.6	408.5	317
100	225	70.4	927.5	354
100	225	70.4	408.5	406
100	75	217.6	927.5	2
100	75	217.6	408.5	2
100	75	70.4	927.5	849
100	75	70.4	408.5	717
100	150	144	668	584
300	150	144	668	119
200	75	144	668	83
200	225	144	668	712
200	150	70.4	668	428
200	150	217.6	668	434
200	150	144	408.5	530
200	150	144	927.5	382

MISSION
OF
ROME LABORATORY

Mission. The mission of Rome Laboratory is to advance the science and technologies of command, control, communications and intelligence and to transition them into systems to meet customer needs. To achieve this, Rome Lab:

    a. Conducts vigorous research, development and test programs in all applicable technologies;

    b. Transitions technology to current and future systems to improve operational capability, readiness, and supportability;

    c. Provides a full range of technical support to Air Force Material Command product centers and other Air Force organizations;

    d. Promotes transfer of technology to the private sector;

    e. Maintains leading edge technological expertise in the areas of surveillance, communications, command and control, intelligence, reliability science, electro-magnetic technology, photonics, signal processing, and computational science.

The thrust areas of technical competence include: Surveillance, Communications, Command and Control, Intelligence, Signal Processing, Computer Science and Technology, Electromagnetic Technology, Photonics and Reliability Sciences.